UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Integração, Controlo e Sequenciamento em Sistemas Robóticos Industriais

António Manuel Pereira Ferrolho

Coimbra – Portugal
2007

UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Integração, Controlo e Sequenciamento em Sistemas Robóticos Industriais

António Manuel Pereira Ferrolho

Coimbra – Portugal
2007

UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Integração, Controlo e Sequenciamento em Sistemas Robóticos Industriais

António Manuel Pereira Ferrolho

Dissertação apresentada a doutoramento em Ciências da Engenharia, área de Engenharia Electrotécnica, na especialidade de Instrumentação e Controlo, na Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Coimbra – Portugal
2007

Tese realizada sob orientação de

Manuel Marques Crisóstomo

Professor Auxiliar do

Departamento de Engenharia Electrotécnica e de Computadores

Faculdade de Ciências e Tecnologia da

UNIVERSIDADE DE COIMBRA

À Aldinha e aos meus filhos Henrique e Ana Rita

Agradecimentos

Quero expressar o meu sincero agradecimento a todos os que, directa ou indirectamente, contribuíram para a realização deste trabalho. Os agradecimentos seriam vastos e numerosos, no entanto, vou referir apenas aqueles que foram os mais importantes.

Em primeiro lugar quero agradecer ao meu orientador, Prof. Doutor Manuel Marques Crisóstomo, pela colaboração, incentivo e orientação prestadas durante a realização do trabalho de doutoramento. Agradeço-lhe, também, a disponibilidade constantemente manifestada nas deslocações feitas a Viseu durante a fase de desenvolvimento da parte prática do trabalho, bem como, mais recentemente, as deslocações feitas à empresa multinacional com a finalidade de acompanhar e orientar a aplicação robótica industrial aí desenvolvida. Obrigado pela confiança que depositou em mim.

Quero ainda agradecer ao Departamento de Engenharia Electrotécnica da Escola Superior de Tecnologia do Instituto Superior Politécnico de Viseu, todas as facilidades concedidas na realização deste trabalho, nomeadamente os recursos disponibilizados para a realização da parte prática do trabalho e os apoios financeiros concedidos para participar em algumas conferências científicas.

Agradeço ao Instituto de Sistemas e Robótica (ISR) do Pólo de Coimbra as verbas concedidas para a minha participação em conferências científicas e publicação de artigos em revistas internacionais.

Finalmente, um agradecimento muito especial a toda a minha família pela compreensão tida durante a realização deste trabalho. Em especial, quero agradecer à minha esposa Aldinha por me ter facilitado e simplificado a vida, tornando menos difícil a execução deste trabalho. Aos meus filhos Henrique e Ana Rita, porque mereciam mais atenção e carinho da minha parte. A eles, peço desculpa pelo tempo que não lhes pude dar. À minha cunhada Sãozinha, agradeço a ajuda e as correcções sugeridas na revisão final do texto da tese.

Por último, estendo os meus agradecimentos a todos aqueles que de uma forma ou de outra contribuíram para a realização deste trabalho e que não foram aqui mencionados.

Resumo

Actualmente, a automatização dos modernos sistemas de produção industriais não é uma tarefa fácil, pois, normalmente, estes sistemas usam equipamentos (*e.g.*, robôs, máquinas CNC, entre outros) de diferentes fabricantes, tendo cada um deles o seu ambiente e linguagem de programação. Assim, a integração e coordenação de robôs e máquinas CNC de diferentes fabricantes nestes sistemas de produção é uma tarefa difícil. A flexibilidade destes modernos sistemas de produção permite um grande número de configurações das estações de trabalho, bem como um conjunto de alternativas para a sequência de operações nas máquinas, tornando a integração, o controlo e o sequenciamento muito complexos.

Neste trabalho de doutoramento foram desenvolvidas várias ferramentas de software e hardware com o intuito de permitirem a integração e o controlo de robôs manipuladores, máquinas CNC industriais, transportadores, motores, sensores, entre outros, nos modernos sistemas de produção.

Foram concebidos novos operadores genéticos que permitem um melhor desempenho dos algoritmos genéticos na resolução de problemas de sequenciamento de *jobs*. Com base nos operadores genéticos concebidos, foram desenvolvidos algoritmos de sequenciamento capazes de resolver problemas reais de sequenciamento. No âmbito deste trabalho, foi concebida a ferramenta de software *hybrid and flexible genetic algorithm* (HybFlexGA) que engloba os vários modelos de sequenciamento desenvolvidos. Desta forma, os resultados computacionais obtidos pelos modelos de sequenciamento propostos demonstram a eficiência dos mesmos, na resolução de problemas reais de sequenciamento.

Para além do trabalho proposto, esta tese fornece uma visão global dos algoritmos genéticos, incluindo, também, as instâncias utilizadas nos problemas de sequenciamento de *jobs* e os resultados dos testes computacionais realizados.

Abstract

Nowadays, the automation of modern industrial production systems is not an easy task, because these systems normally use equipment (e.g., robots, computer numerical control (CNC) machines, and so on) from different manufactures with their own programming language and environments. Thus, it is difficult to integrate and coordinate robots and CNC machines from different manufactures in such production systems. The flexibility of these modern production systems allows a large number of work station configurations, as well as a group of alternatives for the sequence of operations in the machines, making integrating, controlling and scheduling very complex.

In this PhD work several software and hardware tools were developed with the aim of allowing robots, CNC machines, conveyors, sensors, and so on in the modern industrial production systems to be integrated and controlled.

A new concept of genetic operators for scheduling problems was developed to improve the genetic algorithms to resolve job scheduling problems. With these genetic operators we developed scheduling algorithms to solve real scheduling problems. A software tool called hybrid and flexible genetic algorithm (HybFlexGA) was developed to include the several scheduling models presented. The computational results obtained from the proposed scheduling models demonstrate their good performance and efficiency in solving real scheduling problems.

This work also includes a global review of the basic aspects of genetic algorithms, as well as instances used in the job scheduling problems and the computational results obtained.

Índice

Siglas e Abreviaturas

ACL – *Advanced Control Language*

Adj2JC – *Adjacent two-job change*

AG – Algoritmos Genéticos

AGV – *Automated Guided Vehicles*

API – *Application Programming Interface*

Arb20%JC – *Arbitrary 20%-job change*

Arb2JC – *Arbitrary two-job change*

Arb3JC – *Arbitrary three-job change*

ATS – *Advanced Terminal Software*

CAD – *Computer Aided Design*

CAM – *Computer Aided Manufacturing*

CE – Computação Evolucionária

CFF – Célula Flexível de Fabrico

CIM – *Computer Integrated Manufacturing*

CMM – *Coordinate Measuring Machine*

CNC – *Computer Numerical Control*

CPU – *Central Processing Unit*

CX – *Cycle crossover*

DLL – *Dynamic Link Library*

DNC – *Direct Numerical Control*

EE – Estratégias de Evolução

FMC – *Flexible Manufacturing Cells*

FMS – *Flexible Manufacturing Systems*

FT – *Fisher and Thompson*

GA – *Genetic Algorithms*

GUI – *Graphical User Interface*

HybFlexGA – *Hybrid and Flexible Genetic Algorithm*

IEEE – *Institute of Electrical and Electronics Engineers, Inc.*

ITIO – Intervalo dos tempos de início das operações

JIT – *Just-in-Time*

LA – *Lawrence*

MCO – Mecanismo de coordenação das operações

MSA – Mecanismo de sequências alternativas

NC – *Numerical Control*

OPC1C – *One-point crossover: 1 child*

OPC2C – *One-point crossover: 2 children*

OX – *Order crossover*

PBX – *Position based crossover*

PE – Programação Evolucionária

RAD – *Rapid Application Development*

RPC – *Remote Procedure Calls*

SA – *Simulated Annealing*

SB – *Shifting Bottleneck*

SC – *Shift change*

SFF – Sistemas Flexíveis de Fabrico

TCP/IP – *Transmission Control Protocol / Internet Protocol*

TPC1CV1 – *Two-point crossover: 1 child (Version I)*

TPC1CV2 – *Two-point crossover: 1 child (Version II)*

TPC2CV1 – *Two-point crossover: 2 children (Version I)*

TPC2CV2 – *Two-point crossover: 2 children (Version II)*

TPC3CV1 – *Two-point crossover: 3 children (Version I)*

TPC3CV2 – *Two-point crossover: 3 children (Version II)*

TPC4C – *Two-point crossover: 4 children*

TQM – *Total Quality Management*

TS – *Tabu Search*

TSP – *Travelling Salesman Problem*

USB – *Universal Serial Bus*

Nota: Ao longo desta tese são, por vezes, usados termos e frases em Inglês. Esta opção deve-se ao facto de muitas vezes os termos em Inglês serem mais esclarecedores, não havendo muitas vezes o termo correspondente em Português.

Capítulo 1 Introdução

"Quanto mais quero uma coisa feita, menos a considero como trabalho"
Richard Bach

1.1 Introdução

Neste capítulo, apresentam-se os aspectos que motivaram a elaboração desta tese de doutoramento, os objectivos que a orientaram e as contribuições científicas alcançadas. Por último, finalizamos o capítulo com um resumo do conteúdo de cada um dos capítulos integrados neste trabalho.

1.2 Motivação

Nos últimos anos, a automatização dos modernos sistemas de produção industriais, em particular dos sistemas flexíveis de fabrico (SFF) e das células flexíveis de fabrico (CFF), tem sido objecto de investigação. A automatização destes sistemas de produção tem proporcionado um impacto positivo no melhoramento da indústria de manufactura mundial. Sendo inegáveis as vantagens inerentes aos modernos sistemas de produção (*e.g.*, a variedade e a qualidade dos produtos, a capacidade de adaptação a novos produtos, a tolerância a avarias, entre outras), é também indiscutível que a automatização dos mesmos acarreta sérias dificuldades associadas à integração e ao controlo dos equipamentos (*e.g.*, robôs, máquinas CNC, transportadores, entre outros). De facto, o controlo de um moderno sistema de produção envolve a resolução de problemas de complexidade assinalável, sem a qual não será possível atingir o objectivo pretendido. A flexibilidade destes sistemas torna possível um grande número de configurações das estações de trabalho, bem como um conjunto de alternativas para a sequência de operações nas máquinas, tornando a integração, o controlo e o sequenciamento muito complexos.

Os problemas reais de sequenciamento são de resolução muito complexa, sendo conhecidos em termos de teoria da complexidade como NP difíceis (*NP-hard*), para os quais o tempo requerido para a determinação de um plano óptimo ou sub-óptimo aumenta exponencialmente com o tamanho do problema. Nos últimos anos, alguns autores desenvolveram vários algoritmos para a resolução de problemas de sequenciamento, mas os resultados computacionais não são encorajadores. Para pequenos problemas, a técnica *Branch and Bound* tem-se revelado muito boa, enquanto, para problemas de grande dimensão esta técnica é insustentável do ponto de vista computacional.

Os algoritmos genéticos (AG) baseiam-se nos princípios Darwinianos da evolução natural das espécies e podem ser vistos como um método de resolução automática de problemas que obtém soluções através da combinação de outras soluções. Os recentes avanços têm permitido a utilização dos AG na resolução de problemas industriais, nomeadamente ao nível da resolução de problemas de sequenciamento (*scheduling*). Porém, um AG na sua forma básica é insuficiente para a resolução deste tipo de problemas [Syswerda, 1991], [Murata *et al.*, 1994]. Por exemplo, a codificação *bit-string* é inadequada; a selecção através da proporcionalidade do *fitness* conduz à convergência prematura; alguns operadores de cruzamento e mutação tornam as soluções descendentes inviáveis, e ainda, um ajuste inadequado dos parâmetros pode diminuir a performance global do AG. Assim, as decisões a tomar na elaboração de um AG devem ser analisadas cuidadosamente, uma vez que existem muitas possibilidades de implementação.

A motivação deste trabalho resultou de duas necessidades: primeira, desenvolver mecanismos robustos capazes de permitirem a integração e o controlo de vários equipamentos nos modernos sistemas de produção industriais (*e.g.*, robôs manipuladores e máquinas CNC, entre outros); segunda, desenvolver modelos de sequenciamento baseados em algoritmos genéticos para os modernos sistemas de produção.

1.3 Objectivos

Os principais objectivos deste trabalho são enumerados nos pontos seguintes:

- Desenvolver ferramentas de software que permitam a integração e o controlo de robôs manipuladores e máquinas CNC industriais nos modernos sistemas de produção.

- Concepção de hardware e software capazes de permitirem a integração e o controlo de equipamentos de diferentes fabricantes em sistemas de produção, como por exemplo, transportadores, motores, sensores, entre outros.
- Desenvolver uma célula flexível de fabrico com características industriais, com o objectivo de se estudarem estes modernos sistemas de produção e de se testarem as ferramentas de software e hardware desenvolvidas.
- Desenvolver operadores genéticos que permitam melhorar o desempenho dos algoritmos genéticos na resolução de problemas de sequenciamento de *jobs*.
- Desenvolver modelos, com base em algoritmos genéticos, para a resolução de problemas de sequenciamento de *jobs*.
- Conceber uma ferramenta de software que englobe os vários modelos de sequenciamento desenvolvidos no âmbito deste trabalho.

1.4 Contribuições Científicas

A investigação desenvolvida e apresentada nesta tese de doutoramento teve como finalidade atingir os objectivos enunciados na secção anterior. Assim, deste trabalho de doutoramento resultaram várias contribuições científicas, das quais se destacam:

1. O desenvolvimento de várias ferramentas de software que permitem a integração e o controlo de robôs manipuladores e máquinas CNC industriais em modernos sistemas de produção. A concepção de hardware e software capazes de permitirem a integração e o controlo dos mais variados equipamentos de diferentes fabricantes nos modernos sistemas de produção (*e.g.*, transportadores, motores, sensores, entre outros). Estas ferramentas de software e hardware foram utilizadas e testadas numa célula flexível de fabrico, com o objectivo de demonstrar a operacionalidade e utilidade das mesmas. De referir que, uma das ferramentas de software desenvolvida para os robôs manipuladores industriais encontra-se actualmente em utilização intensiva numa empresa multinacional do ramo automóvel.
2. A concepção de novos operadores genéticos para utilização em problemas de sequenciamento de *jobs*.
3. O desenvolvimento de um algoritmo genético capaz de resolver problemas reais de sequenciamento com uma só máquina.
4. A construção de um novo modelo para a resolução de problemas de sequenciamento do tipo *Job-Shop*.

5. A concepção do software *hybrid and flexible genetic algorithm* (HybFlexGA) que reúne os vários modelos de sequenciamento desenvolvidos.

Deste trabalho de doutoramento resultaram já várias publicações em revistas, livros e conferências internacionais:

Revistas Internacionais

1. António Ferrolho e Manuel Crisóstomo, "Intelligent Control and Integration Software for Flexible Manufacturing Cells", IEEE Transactions on Industrial Informatics, Vol. 3, no. 1, ISSN: 1551-3203, pp. 3-11, February 2007.
2. António Ferrolho e Manuel Crisóstomo, "Scheduling Jobs in Flexible Manufacturing Cells with Genetic Algorithms", Research in Computing Science, Special issue: Advances in Computer Science and Engineering, Vol. 23, ISSN: 1870-4069, 2006, pp. 31-40.
3. António Ferrolho, Manuel Crisóstomo e Miguel Lima, "Scheduling in Flexible Manufacturing Cells", International Journal of Factory Automation, Robotics and Soft Computing, Issue 2, April 2006, ISSN: 1828-6984, pp. 56-62.
4. António Ferrolho e Manuel Crisóstomo, "Scheduling and Control of Flexible Manufacturing Cells Using Genetic Algorithms", WSEAS Transactions on Computers. Issue 6, Volume 4, June 2005, ISSN: 1109-2750, pp. 502-510.
5. António Ferrolho e Manuel Crisóstomo, "Genetic Algorithms: concepts, techniques and applications", WSEAS Transactions on Advances in Engineering Education. Issue 1, Volume 2, January 2005, ISSN: 1790-1979, pp. 12-19.

Capítulos de Livros

1. António Ferrolho e Manuel Crisóstomo, "Control and Scheduling Software for Flexible Manufacturing Cells", Industrial Robotics: Programming, Simulation and Applications, ISBN: 3-86611-286-6, pp. 315-340, Advanced Robotic Systems, 2007.

Conferências Internacionais

1. António Ferrolho e Manuel Crisóstomo, "Single Machine Total Weighted Tardiness Problem with Genetic Algorithms", Proceedings of the IEEE International Conference on Computer Systems and Applications (AICCSA07), Amman, Jordania, May 13-16, 2007, em CD-ROM.

2. António Ferrolho e Manuel Crisóstomo, "Genetic Algorithm for the Single Machine Total Weighted Tardiness Problem", Proceedings of the IEEE International Conference on E-Learning in Industrial Electronics (ICELIE06), Hammamet, Tunisia, December 18-20, 2006, ISBN: 1-4244-0324-3, pp. 17-22.
3. António Ferrolho e Manuel Crisóstomo, "Control and Scheduling in Flexible Manufacturing Cells", Proceedings of the IEEE International Conference on Industrial Technology (ICIT06), Mumbai, India, December 15-17, 2006, ISBN: 1-4244-0726-5, pp. 1241-1246.
4. António Ferrolho e Manuel Crisóstomo, "A New Concept of Genetic Operators for Scheduling Problems", Proceedings of the IEEE 4th International Conference on Computational Cybernetics, Tallinn, Estonia, August 20-22, 2006, pp. 131-136.
5. António Ferrolho, Manuel Crisóstomo e Miguel Lima, "Intelligent Control Software for Industrial CNC Machines", Proceedings of the IEEE 9th International Conference on Intelligent Engineering Systems, Cruising on Mediterranean Sea, September 2005, em CD-ROM.
6. António Ferrolho e Manuel Crisóstomo, "Flexible Manufacturing Cell: Development, Coordination, Integration and Control", Proceedings of the IEEE 5th International Conference on Control and Automation, Budapest, Hungary, June 2005, pp. 1050-1055.
7. António Ferrolho e Manuel Crisóstomo, "Genetic Algorithms for Solving Scheduling Problems in Flexible Manufacturing Cells", Proceedings of the 4th WSEAS International Conference on Electronics, Signal Processing and Control (ESPOCO2005), Rio de Janeiro, Brazil, April 2005, em CD-ROM.
8. António Ferrolho e Manuel Crisóstomo, "Development of a Flexible Manufacturing Cell", Proceedings of the 3th WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA2004), Salzburg, Austria, February 2004, em CD-ROM.
9. António Ferrolho e Manuel Crisóstomo, "Software Development to Control the Scorbot ER VII Robot With a PC", Proceedings of the 5th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering (MMACTEE 2003)", Vouliagmeni, Athens, Greece, December 2003, em CD-ROM.

Para além das publicações referidas anteriormente, encontram-se em processo de revisão os seguintes artigos:

1. António Ferrolho, Manuel Crisóstomo e Robert Wójcik, "Job Shop Scheduling Problems with Genetic Algorithms", submetido a uma conferência internacional.
2. António Ferrolho e Manuel Crisóstomo, "A Hybrid and Flexible Genetic Algorithm for the Job-Shop Scheduling Problem", submetido a uma conferência internacional.
3. António Ferrolho e Manuel Crisóstomo, "Optimization of Genetic Operators for Scheduling Problems", submetido a uma revista internacional.
4. António Ferrolho e Manuel Crisóstomo, "Scheduling Jobs with Genetic Algorithms", submetido a uma revista internacional.

1.5 Estrutura da Dissertação

Este capítulo introdutório apresentou a motivação que esteve na origem deste trabalho, os objectivos e as contribuições científicas resultantes da investigação desenvolvida.

No capítulo 2, *algoritmos genéticos*, é apresentada uma visão geral sobre os algoritmos genéticos, uma das tecnologias mais em voga nos últimos anos. É feito o enquadramento dos algoritmos genéticos dentro das técnicas de pesquisa meta-heurísticas, e também, são apresentados os conceitos fundamentais dos algoritmos genéticos. Por último, são apresentados dois exemplos com a finalidade de se explicar melhor algumas das características dos algoritmos genéticos.

O capítulo 3, *robôs manipuladores e máquinas CNC industriais: controlo e software distribuído*, descreve o desenvolvimento de várias ferramentas de software para utilização em robôs e máquinas CNC industriais. O grande objectivo destas ferramentas de software é permitir a integração e o controlo destes equipamentos, de uma maneira fácil e eficiente, nos modernos sistemas de produção.

O capítulo 4, *célula flexível de fabrico: desenvolvimento, coordenação, integração e controlo*, descreve o desenvolvimento de uma célula flexível de fabrico com características industriais, com o objectivo de se estudarem estes sistemas de produção e de se testarem as ferramentas de software desenvolvidas no âmbito do capítulo 3.

No capítulo 5, *sequenciamento de tarefas na célula flexível de fabrico*, é abordado o problema do sequenciamento de tarefas (*jobs*) em células flexíveis de fabrico. Inicialmente, é apresentado o

problema de sequenciamento, a terminologia e os conceitos utilizados ao longo da tese, e por último, apresenta-se a arquitectura do software desenvolvida e implementada na célula flexível de fabrico, bem como os testes computacionais realizados.

No capítulo 6, *algoritmos genéticos aplicados aos problemas de sequenciamento com uma só máquina*, é apresentado um novo conceito de operadores genéticos para os problemas de sequenciamento de *jobs*. É desenvolvido o software *hybrid and flexible genetic algorithm* (HybFlexGA), usado para examinar a performance dos vários operadores de cruzamento e de mutação. São, também, apresentados os resultados computacionais obtidos em problemas reais de sequenciamento com uma só máquina.

No capítulo 7, *algoritmos genéticos aplicados aos problemas de sequenciamento com duas ou mais máquina*, são abordados os problemas de sequenciamento do tipo *Job-Shop* e é proposto um modelo de sequenciamento para a resolução deste tipo de problemas. São apresentados três exemplos de sequenciamento *Job-Shop* com o objectivo de ilustrar e explicar o funcionamento do algoritmo de sequenciamento proposto. Por último, são apresentados os testes e resultados computacionais obtidos pelo modelo de sequenciamento proposto e por vários algoritmos desenvolvidos por outros autores.

No capítulo 8, *conclusões*, são apresentadas as conclusões relativas ao trabalho desenvolvido e enumeram-se algumas perspectivas para desenvolvimento futuro.

Para além dos capítulos referidos, a tese é ainda composta por dois apêndices. No apêndice A, encontra-se a população inicial e as instâncias utilizadas nos testes computacionais do capítulo 6. No apêndice B, estão as instâncias dos problemas de sequenciamento *Job-Shop* utilizadas nos testes computacionais do capítulo 7.

Capítulo 2 Algoritmos Genéticos

"The most incomprehensible thing about the world is that it is comprehensible."

Albert Einstein

2.1 Introdução

Este capítulo pretende dar uma visão geral sobre os algoritmos genéticos (AG), uma das tecnologias mais em voga nos últimos anos. Os AG baseiam-se nos princípios Darwinianos da evolução natural das espécies. Todas as espécies existentes na Terra resultam de uma evolução ao longo de milhões de anos, em que cada espécie evoluiu de modo a adaptar-se melhor ao ambiente, procurando tirar melhor partido das suas características inatas, para sobreviver. A evolução natural é responsável pelo aparecimento de todos os organismos vivos no nosso planeta tais como os conhecemos actualmente. É um processo muito lento que ocorre há milhões de anos e que continua neste preciso momento. É esse processo de adaptação das espécies de organismos vivos que lhes permite sobreviver num ambiente sujeito a mudanças regulares. Por vezes, algumas espécies desaparecem por falta de adaptação a esse meio ambiente. Acerca de algumas dezenas de anos atrás surgiu o interesse pelo estudo da possibilidade de simular computacionalmente o processo da evolução natural e de usar estas simulações como ferramentas de optimização.

John H. Holland, investigador da Universidade de Michigan, propôs a construção de um algoritmo matemático para a optimização de sistemas complexos, ao qual chamou de AG. O AG, constituído por cromossomas artificiais, foi estruturado de forma a que as informações referentes a um determinado sistema pudessem ser codificadas de uma maneira análoga aos cromossomas biológicos. Em 1975, John Holland publicou o livro "Adaptation in Natural and Artificial Systems", onde apresenta e explica a teoria dos AG [Holland, 1975]. Nos anos 80, David Goldberg, aluno de John Holland, aplicou os AG, pela primeira vez, à resolução de problemas industriais [Goldberg, 1989]. Desde então para cá, os AG têm sido aplicados aos mais diversos tipos de problemas.

Neste capítulo pretende-se dar uma visão global sobre os AG quando aplicados à optimização de problemas, atendendo à natureza do trabalho desenvolvido nos capítulos 5, 6 e 7 desta dissertação. Assim, este capítulo está estruturado da seguinte forma: na secção 2.2 encontra-se um resumo das técnicas de pesquisa meta-heurísticas; na secção 2.3 é apresentada uma analogia entre a evolução natural e os algoritmos genéticos; na secção 2.4 mostramos o funcionamento geral de um AG; a secção 2.5 faz uma breve introdução aos aspectos práticos relacionados com a implementação dos AG; na secção 2.6 são utilizados dois exemplos com o objectivo de explicarmos melhor algumas características dos AG e, por último, a secção 2.7 faz um resumo deste capítulo.

2.2 Técnicas de pesquisa meta-heurísticas

Como mostra a figura 2.1, os AG são uma das técnicas de pesquisa meta-heurísticas inspiradas na natureza. Dentro da computação evolucionária (CE), existem três ramos principais de estudo: programação evolucionária (PE) [Fogel *et al.*, 1966], estratégias de evolução (EE) [Rechenberg, 1973] e algoritmos genéticos (AG) [Holland, 1975].

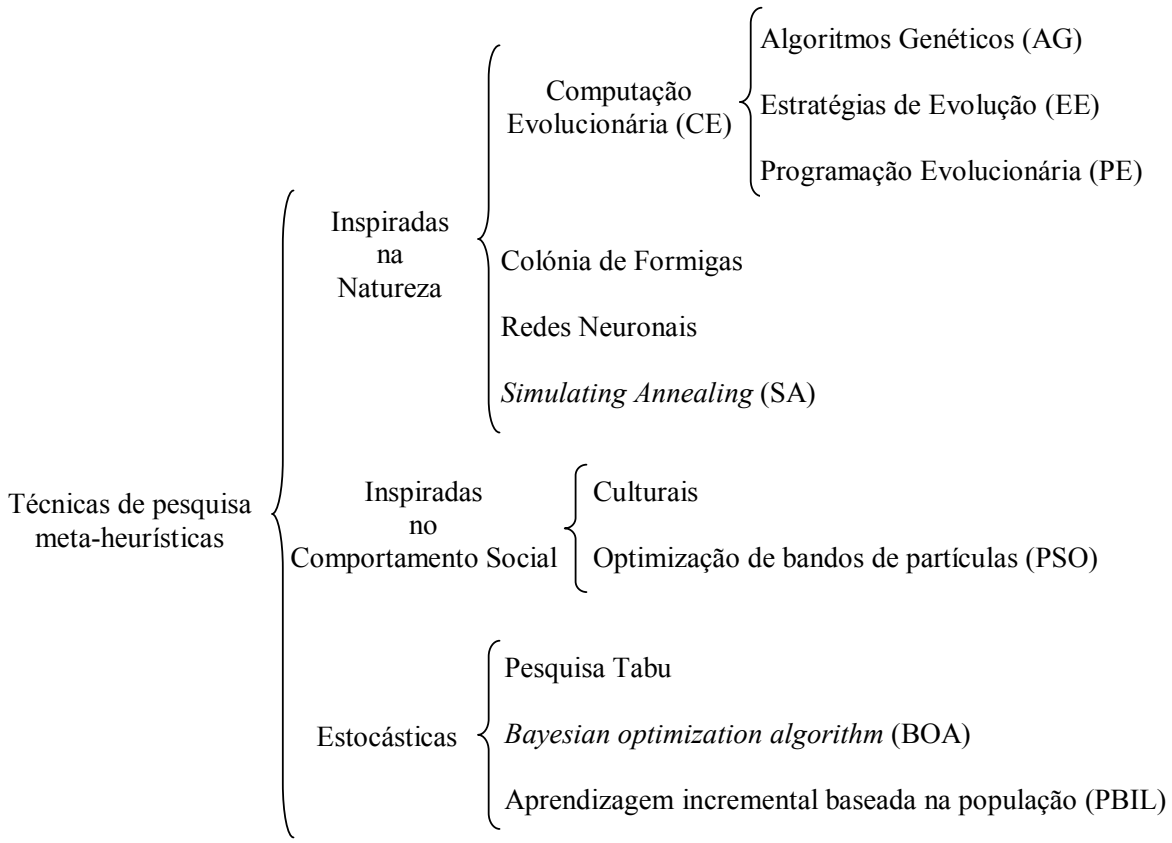


Figura 2.1 Técnicas de pesquisa meta-heurísticas.

Os algoritmos baseados na computação evolucionária (ver figura 2.1) são estocásticos e iterativos, não garantindo a convergência para a melhor solução. A conclusão do processo iterativo pode ser obtida quando se atinge um número máximo de gerações pré-definido ou obtendo uma solução aceitável. Estes algoritmos operam sobre um conjunto de indivíduos (população), em que cada indivíduo representa uma potencial solução para o problema em estudo. Esta solução é obtida por meio de um mecanismo de codificação e descodificação. Inicialmente, a população é gerada aleatoriamente e é atribuído um valor a cada indivíduo na população, através de uma função de aptidão. Este valor, conhecido por valor de aptidão (na literatura inglesa o valor de aptidão é designado por *fitness*), é uma medida da sua qualidade relativamente ao problema considerado, sendo usado para orientar a pesquisa.

Os princípios básicos dos AG foram estabelecidos em [Holland, 1975], no entanto, actualmente o algoritmo original proposto por Holland é designado por AG canónico ou AG simples [Goldberg, 1989]. Estes algoritmos serão descritos com detalhe ao longo deste capítulo.

As estratégias de evolução (EE) foram inicialmente introduzidas por [Rechenberg, 1973], tendo sido alvo de posteriores desenvolvimentos em [Schwefel, 1981]. A sua primeira versão considera apenas dois indivíduos, um progenitor e um descendente, e utilizam uma representação decimal para a sua codificação. As EE consideram, ainda, um operador de mutação e um mecanismo de selecção.

A programação evolucionária (PE), originalmente introduzida por [Fogel *et al.*, 1966], apresenta algumas características comuns com os AG. A diferença fundamental, em termos da sua funcionalidade, relaciona-se com a forma como os descendentes são gerados. Não existe qualquer preocupação em imitar os operadores naturais, sendo o operador de mutação o único responsável pela geração de descendentes. Como na PE não existe um operador de recombinação, ao contrário do que acontece com os AG, é possível usar um qualquer tipo de representação desde que se defina um operador de mutação adequado.

Relativamente às restantes técnicas meta-heurísticas que se encontram na figura 2.1, estas vão para além do âmbito deste capítulo e, por conseguinte, não são aqui apresentadas. No entanto, uma descrição detalhada sobre essas técnicas de pesquisa meta-heurísticas pode ser encontrada em [Figueiredo, 2002] e [Pires, 2005].

2.3 Os Algoritmos Genéticos e a Biologia

Os AG constituem um conjunto de métodos adaptativos que podem ser usados na resolução de problemas de pesquisa e optimização. Estes métodos pretendem simular os processos que são essenciais à evolução natural de populações que, em termos biológicos, são constituídas por organismos vivos. Ao longo de muitas gerações, populações de organismos vivos evoluem de acordo com os princípios da selecção natural, que foram pela primeira vez descritos por Darwin na obra "Sobre a Origem das Espécies por meio da Selecção Natural", publicada em 1859.

Na natureza, os indivíduos de uma população competem entre si na obtenção de recursos tais como: alimentação, água, abrigo, entre outros. Os membros da mesma espécie também competem frequentemente para atrair um companheiro para reprodução. Os indivíduos com maior capacidade de sobrevivência e de atracção de companheiros para reprodução são os que terão mais probabilidades de virem a ter um maior número de descendentes. Os menos capazes terão um menor número de descendentes. Desta forma, os genes dos indivíduos mais bem adaptados ao meio ambiente serão espalhados por um crescente número de indivíduos a cada sucessiva geração. A combinação de boas características de diferentes antepassados pode em alguns casos produzir descendentes bem adaptados ao meio ambiente, cujo grau de adaptação é potencialmente melhor que o dos progenitores. Com este processo, e ao longo de sucessivas gerações, uma dada espécie evolui no sentido de ter cada vez mais indivíduos melhor adaptados ao seu ambiente.

Como mostra a tabela 2.1, os AG usam uma analogia directa com o processo de evolução natural. Estes trabalham com uma população de soluções (indivíduos) do problema considerado, e a cada indivíduo é atribuído um valor de aptidão (ou *fitness*) que corresponde à sua capacidade de sobrevivência e reprodução. Os indivíduos com os maiores valores de aptidão possuem maiores oportunidades de se combinarem com outros indivíduos da população. Na reprodução são gerados novos indivíduos que herdam algumas características de cada um dos progenitores. A combinação de boas características dos progenitores permite a geração de indivíduos (soluções) melhores.

Tabela 2.1 Analogia entre a evolução natural e os algoritmos genéticos

Evolução Natural	Algoritmos Genéticos
Cromossoma, indivíduo	*String*, vector, solução
Gene	Carácter, parâmetro
Alelo	Valor

Lugar (*locus*)	Posição no vector
Reprodução sexual	Operador de cruzamento
Mutação	Operador de mutação
População	Conjunto de soluções
Gerações	Iterações, ciclos

2.4 Funcionamento Geral de um Algoritmo Genético

Um AG trabalha sobre uma população de indivíduos cujo tamanho permanece, em geral, constante ao longo das gerações. Estes indivíduos, designados cromossomas, representam as soluções candidatas para um dado problema. Os cromossomas são constituídos por um conjunto de genes (*e.g.*, uma sequência de *bits*), que podem tomar diferentes valores (*e.g.*, 0 ou 1). Aos diferentes valores que um gene pode tomar dá-se o nome de alelo. A figura 2.2 apresenta um exemplo de um cromossoma artificial (cromossoma binário), em que cada célula representa um gene e os alelos são os valores 1 e 0.

1	0	0	1	1	1	1

Figura 2.2 Cromossoma artificial.

O funcionamento de um AG pode descrever-se como a seguir se indica. A população inicial é, normalmente, gerada aleatoriamente. Como cada indivíduo da população representa uma solução candidata para o problema, teremos várias soluções possíveis geradas aleatoriamente que ficaram distribuídas no espaço de procura. Esta população inicial evoluirá ao longo de um número de gerações através da actuação de mecanismos de selecção, recombinação e mutação. Os melhores indivíduos da população são seleccionados de acordo com a sua qualidade (medida pela função de avaliação) para que se possam reproduzir. Através da aplicação de operadores de cruzamento, a reprodução combina as características de dois indivíduos (progenitores) de modo a gerar dois novos indivíduos (descendentes), obtidos por composição do material genético dos seus progenitores. Estes novos indivíduos poderão ainda ser alterados através dos operadores de mutação. Estes operadores garantem a existência de alguma diversidade genética na população, prevenindo a convergência prematura para óptimos locais. A forma como se define a nova população varia de acordo com as técnicas de selecção utilizadas no AG.

Estes passos repetem-se até um dado critério de paragem ser atingido (por exemplo, um número predefinido de gerações). Ao longo das gerações, a população sofrerá um processo evolutivo que, em princípio, conduzirá à solução óptima, sendo essa a solução que o algoritmo devolve.

Sucintamente, um AG é um processo iterativo que durante a iteração t (denominada geração) mantém constante uma população de soluções potenciais (cromossomas) $P(t) = \{x_t^1, x_t^2, ..., x_t^n\}$. Cada solução x_t^i é avaliada para fornecer uma medida da sua aptidão (qualidade da solução). De seguida, uma nova geração $t+1$ é formada, seleccionando os indivíduos da geração anterior. A conclusão do processo iterativo pode ser obtida quando se atinge um número máximo de iterações, um limite temporal ou uma solução aceitável. Após a escolha dos requisitos necessários para a implementação do AG, a estrutura de um AG simples é fornecida pela figura 2.3, onde $P(t)$ é a população de cromossomas na iteração t.

início
    t=0;
    inicializar aleatoriamente $P(t)$;
    avaliação $P(t)$;
    reptir
        cruzamento $P(t)$;
        mutação $P(t)$;
        avaliação $P(t)$;
        obter $P(t+1)$ a partir de $P(t)$;
        t=t+1;
    até se verificar a condição de conclusão
fim

Figura 2.3 Algoritmo genético simples.

Existem vários tipos de selecção e diferentes operadores genéticos que são normalmente utilizados, alguns dos quais serão descritos mais adiante. A escolha do operador de cruzamento é muito importante, visto que ele permite ao AG explorar globalmente o espaço de procura e encaminhar-se para regiões mais promissoras, aproximando-se assim da melhor solução do problema. Por outro lado, o operador de mutação altera aleatoriamente o valor de um ou mais genes do cromossoma, impedindo a convergência prematura do AG para máximos locais.

2.5 Elementos Básicos de um Algoritmo Genético

Esta secção destina-se a apresentar uma breve introdução aos aspectos práticos relacionados com a implementação dos AG, embora uma descrição mais completa, incluindo os fundamentos teóricos, possa ser encontrada em muitos outros textos [Goldberg, 1989], [Davis, 1991], [Fogel, 1995], [Michalewicz, 1996], [Mitchell, 1996], [Back, 1996], [Banzhaf *et al.*, 1998], [Goldberg, 2002] e [Eiben e Smith, 2003].

Como já foi referido, um AG permite a procura de soluções em espaços complexos, normalmente intratáveis pelos métodos tradicionais. Para que o algoritmo permita a resolução do problema em causa, é necessário que este codifique correctamente as especificações do problema a resolver. Assim, é necessário ter em conta os seguintes aspectos:

- Codificação a utilizar nos cromossomas da população;
- Função de avaliação que permite medir o mérito de cada cromossoma;
- Método de selecção a utilizar garantindo que os melhores cromossomas da população tenham maior probabilidade de sobreviver e de se reproduzir;
- Operadores a aplicar para a obtenção de novos cromossomas e as probabilidades com que estes actuarão;
- Características da população, como por exemplo: tamanho, geração da população inicial, entre outras.

2.5.1 Codificação dos cromossomas

A escolha da codificação dos cromossomas é a primeira componente a ser decidida na implementação de um AG. A codificação no AG canónico é binária e a dimensão do cromossoma é dada pelo número de *bits* que o constituem. A codificação e a complexidade da descodificação dependem do problema que se pretende resolver. Por exemplo, na implementação de um AG para resolver o problema do caixeiro-viajante (TSP – *Travelling Salesman Problem*) [Fogel, 1998] e [Cormen, 1990] com 32 cidades, o cromossoma pode ser representado por 160 *bits*. Cada cidade pode ser representada por cinco *bits*, uma vez que $2^5 = 32$. A codificação do cromossoma deverá ser uma sequência que contenha todas as cidades, logo 5x32 = 160 *bits*. Neste caso, a descodificação é simples já que cada conjunto de cinco *bits* representa uma cidade. A manipulação dos dados poderá ser feita depois de uma conversão para decimal, obtendo-se uma sequência de valores que representa o caminho codificado no cromossoma. No entanto, se o número de cidades for muito superior poderá ser necessário codificar a informação de outra forma, para diminuir a dimensão dos cromossomas. Nestes casos a descodificação pode ser mais complexa e, inevitavelmente, o tempo de execução do AG aumenta.

A codificação de cromossomas que tem vindo a ser usada com maior frequência é a codificação binária. No entanto, verifica-se uma crescente falta de unanimidade quanto ao facto desta codificação ser a mais adequada e natural. Um exemplo óbvio, quanto à sua falta de adequação, relaciona-se com a resolução de problemas que necessitam de valores com uma elevada precisão. Estes problemas requerem a utilização de um elevado número de *bits* o que obriga a um maior

esforço computacional para explorar um vasto espaço de potenciais soluções. Assim, a codificação mais adequada para estes problemas passa pela utilização de números reais para representar as soluções (cromossomas) [Michalewicz, 1996].

Em suma, a codificação dos cromossomas varia de problema para problema. Portanto, a escolha da codificação mais adequada ao problema que se pretende resolver é um aspecto muito importante, do qual dependerá o desempenho do AG [Ferrolho e Crisóstomo, 2005d].

2.5.2 Função de Avaliação

A finalidade da função de avaliação é fornecer uma medida de qualidade de cada indivíduo (ou solução). A escolha desta função de avaliação depende do problema a resolver. Se nalguns casos a escolha desta função é bastante fácil, noutros, que envolvam restrições de algum tipo, a escolha tem que ser feita com algum cuidado. Esta função, que efectua a avaliação de modo a atribuir um valor de aptidão às potenciais soluções, precisa de ser definida para cada problema que se pretende solucionar. Para um dado cromossoma, a função de avaliação retorna um valor numérico que espelha o mérito do cromossoma para um dado problema.

Suponhamos, como exemplo, que se pretendia desenvolver um AG para optimizar a função matemática (2.1) [Michalewicz, 1996]. Neste caso, a função de avaliação pode ser a própria função matemática.

$$f(x) = x.sen(10\Pi.x) + 1.0 \tag{2.1}$$

2.5.3 Métodos de Selecção

Os métodos de selecção são usados na escolha dos progenitores que irão gerar descendentes e, também, na escolha dos indivíduos que devem passar para a geração seguinte. Quando se implementa um AG, é necessário definir a forma como será realizada a selecção dos indivíduos que vão dar origem à nova geração. Tradicionalmente, o mecanismo de selecção deve permitir que os melhores indivíduos se reproduzam mais vezes, para que, desta forma, a população vá evoluindo para a convergência. Uma selecção muito exigente faz com que a população seja dominada muito rapidamente pelos melhores indivíduos, podendo levar o AG à estagnação num máximo local. Por outro lado, uma selecção pouco exigente poderá conduzir a um processo de evolução muito lento.

Vários métodos de selecção têm sido propostos por vários autores, não existindo, no entanto, uma resposta clara para a escolha do melhor método a utilizar numa dada situação. Assim, a seguir são apresentados os principais métodos de selecção.

Selecção através de roleta

Este método de selecção, também designado por método de amostragem estocástica, foi utilizado no trabalho original de John Holland [Holland, 1975]. Este método baseia-se no valor atribuído a cada indivíduo pela função de avaliação e na qualidade de toda a população. De acordo com a qualidade de cada indivíduo, atribui-se uma parte de um círculo (roleta) a cada um deles. A figura 2.4 mostra uma roleta com 10 indivíduos. Rolando a roleta é seleccionado o individuo cujo arco de círculo se encontra em frente da seta, neste caso o 9.

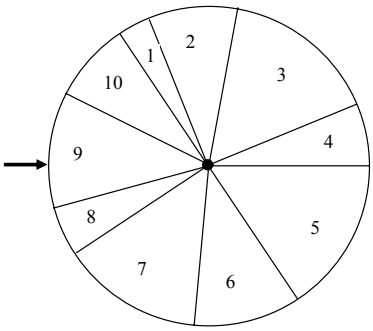


Figura 2.4 Roleta.

A probabilidade de um indivíduo ser seleccionado (porção da roleta atribuída a cada indivíduo) é calculada pela equação (2.2). Assim, o tamanho da zona da roleta atribuída ao individuo x_i depende da sua qualidade (calculada por $f(x_i)$) e da qualidade de todos os indivíduos da população.

$$Porção(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N_{pop}} f(x_i)} \tag{2.2}$$

Selecção de Boltzmann

Na selecção de *Boltzmann*, um valor de aptidão é afectado a um indivíduo de acordo com a distribuição probabilística de *Boltzmann* (2.3), onde o parâmetro T_k, é análogo com a temperatura no processo de arrefecimento de metais. Com o valor de T_k grande, todos os indivíduos têm praticamente a mesma probabilidade de serem seleccionados, mas à medida que T_k diminui, a selecção das boas soluções aumenta em detrimento das restantes.

$$\frac{1}{1+e^{\frac{f(x_i)}{T_k}}} \tag{2.3}$$

Selecção por torneio

Neste método de selecção escolhem-se aleatoriamente dois indivíduos da população. De seguida, gera-se um número aleatório, *r*, entre 0 e 1. Se $r<k$ (*k* é um parâmetro entre 0 e 1), o melhor dos dois indivíduos é seleccionado para reprodução. Caso contrário, selecciona-se o pior individuo. Os dois cromossomas são devolvidos à população e podem voltar a ser escolhidos, visto que este processo é repetido um número de vezes igual ao tamanho da população.

Selecção elitista

Este método de selecção tem como finalidade evitar que os indivíduos de melhor qualidade se percam, retendo um determinado número dos melhores indivíduos para a geração seguinte. Os restantes indivíduos são obtidos através de um outro método de selecção, actuando sobre toda a população inicial.

2.5.4 Operador de cruzamento

O operador de cruzamento é o principal operador utilizado nos AG. Dois indivíduos progenitores são escolhidos da população actual, por um método de selecção, para produzirem dois novos indivíduos, denominados descendentes. Tal como no cruzamento biológico, um descendente herda genes de ambos os progenitores.

Os operadores de cruzamento mais usados são do tipo:

- Cruzamento de ponto único – é seleccionado aleatoriamente um ponto de corte, criando quatro sequências que aparecerão cruzadas nos descendentes. Estes recebem uma sequência de cada um dos progenitores (ver figura 2.5);
- Cruzamento de dois pontos – selecciona aleatoriamente dois pontos de corte, criando seis sequências que aparecerão cruzadas nos descendentes. Neste caso, os descendentes recebem uma sequência de um dos progenitores e duas do outro (ver figura 2.6);
- Cruzamento uniforme – usa uma máscara binária, gerada aleatoriamente, de comprimento igual ao dos cromossomas. Os genes herdados por cada um dos descendentes dependem desta máscara (ver figura 2.7).

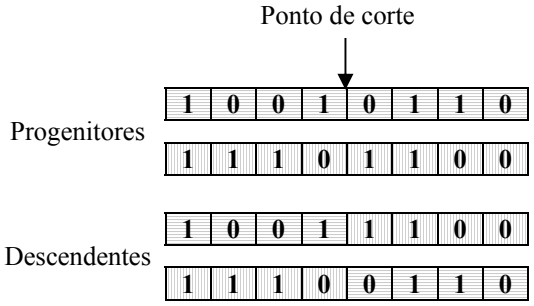


Figura 2.5 Cruzamento de ponto único.

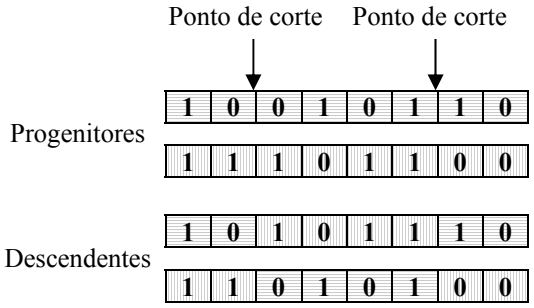


Figura 2.6 Cruzamento de dois pontos.

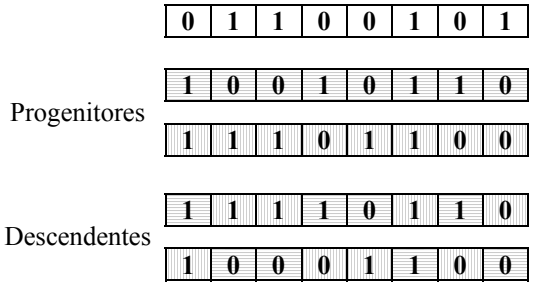


Figura 2.7 Cruzamento uniforme.

Estes operadores foram originalmente desenvolvidos para cromossomas binários [Holland, 1975] e [Goldberg, 1989], embora também possam ser usados em cromossomas que utilizam um alfabeto de símbolos diferentes. Uma descrição mais detalhada sobre os operadores de cruzamento apresentados anteriormente pode ser encontrada em [Michalewicz, 1996], [Madureira, 2003], [Pires, 2005] e [Ferrolho e Crisóstomo, 2005d].

Em certos problemas, os descendentes são sempre soluções viáveis, ou seja, são garantidamente soluções possíveis. Contudo, pode acontecer que os descendentes gerados pelos métodos de cruzamento indicados anteriormente não sejam viáveis. Nesses casos, podem ser utilizados os seguintes operadores de cruzamento: *order crossover* (OX), *cycle crossover* (CX) e *position based crossover* (PBX). Uma descrição mais detalhada sobre estes operadores de cruzamento pode ser encontrada em [Goldberg, 1989], [Michalewicz, 1996] e [Ferrolho e Crisóstomo, 2005d].

O operador de cruzamento é considerado como o verdadeiro responsável pela evolução da qualidade das populações de cromossomas. O facto de recombinar os *bits* dos melhores cromossomas implica que as melhores sequências genéticas da população se propaguem criando novos cromossomas que exploram o espaço de procura em direcção às melhores soluções. Este processo corre, no entanto, o risco de ficar preso em máximos locais. Como veremos a seguir, outras operações são necessárias para introduzir diversidade na população.

2.5.5 Operador de mutação

A mutação tem como objectivo assegurar a diversidade genética de uma população de cromossomas [Holland, 1975], [Goldberg, 1989] e [Davis, 1991]. Usando apenas cruzamentos pode chegar-se a uma situação em que um dos *bits* dos cromossomas tem o mesmo valor em toda a população. Por exemplo, se a determinada altura todos os cromossomas da população tiverem o primeiro *bit* a 1 e a melhor solução tiver necessariamente o primeiro *bit* a 0 o AG nunca atingirá essa solução, se apenas forem utilizados cruzamentos. Esse problema pode ser resolvido com a mutação que, normalmente com uma probabilidade muito baixa, altera o valor de um *bit*. Este operador evita perdas irreparáveis de sequências genéticas. Um valor muito alto da mutação pode, por outro lado, danificar boas sequências genéticas e tornar o AG numa procura quase aleatória.

Apesar de ser visto como um operador secundário em relação ao operador de cruzamento, o operador de mutação desempenha um papel importante no funcionamento do AG. Este operador evita que o algoritmo estagne num máximo local, "agitando-o" e fazendo com que este explore outras regiões eventualmente mais promissoras. Por exemplo, quando são utilizados cromossomas binários, o operador de mutação escolhe aleatoriamente uma posição do cromossoma e altera-o de acordo com o procedimento ilustrado na figura 2.8.

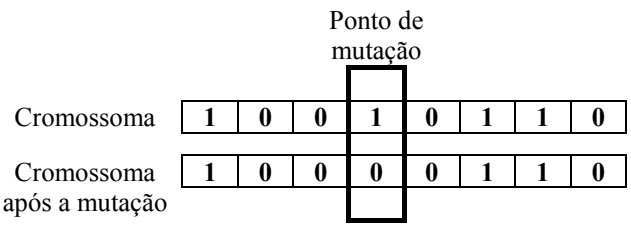


Figura 2.8 Uma simples operação de mutação.

Em suma, a mutação é uma operação importante pelo facto de poder gerar indivíduos com melhores aptidões, fugindo desse modo aos máximos locais do problema. Dito de outra forma, a mutação introduz um factor adicional de diversidade na população.

2.5.6 Parametrização de um Algoritmo Genético

Um AG depende essencialmente de um conjunto de parâmetros para um bom funcionamento. Os principais parâmetros de um AG são: o tamanho da população, a taxa de cruzamento, a taxa de mutação, a taxa de substituição e os critérios de paragem.

Tamanho da População

O tamanho da população é um parâmetro de grande importância para qualquer AG. O seu valor afecta quer a qualidade da solução final quer o tempo de processamento. Uma população pequena apresenta uma diversidade genética pobre em termos de diversidade dos seus elementos constituintes, proporcionando uma menor cobertura do espaço de soluções, pelo que as soluções geradas tendem a ser fracas. Por outro lado, com o aumento do tamanho da população obtém-se um aumento da probabilidade de produzir melhores soluções, através duma maior cobertura do espaço de soluções e prevenindo a convergência prematura, embora à custa de um maior esforço computacional.

Geralmente, os AG admitem que a população se mantém constante, embora nada impeça a sua variação dinâmica. Mas, em termos práticos deve-se ter o cuidado de evitar a diminuição ou crescimento abrupto da população.

Taxa de Cruzamento

A taxa de cruzamento define-se como a medida da possibilidade de aplicação do operador de cruzamento a um dado par de indivíduos. Os valores típicos para esta taxa situam-se no intervalo de 60% a 100%. Quanto maior for esta taxa, maior é a quantidade de indivíduos introduzidos na população. Sendo o tamanho da população normalmente fixo, mais indivíduos tenderão a ser substituídos, logo pode haver a tendência para a perda de indivíduos com aptidão elevada. Para valores baixos desta taxa, gerar-se-ão menos indivíduos em cada geração, o que pode originar um aumento do número de gerações para obter os mesmos resultados.

Taxa de Mutação

Uma taxa de mutação muito elevada conduz a uma pesquisa aleatória que poderá não originar convergência, visto que o mais provável numa mutação é até gerar indivíduos com pior aptidão. Os valores típicos da taxa de mutação são bastante reduzidos (por exemplo, cerca de 1%).

Taxa de Substituição

A taxa de substituição define a percentagem de indivíduos da população que será substituída em cada geração. Se a percentagem de indivíduos a substituir for de 100% todos os indivíduos da população actual são substituídos pelos novos indivíduos resultantes da reprodução. Quanto menor for o valor desta taxa, menor será a diferenciação genética entre gerações e, deste modo, existirá uma convergência mais lenta do algoritmo.

Critérios de Paragem

O critério para a paragem do AG depende do problema em causa e do esforço computacional que é exigido. Em face do tempo e dos recursos disponíveis, é necessário definir qual a qualidade da solução que se pretende. Geralmente adopta-se um dos seguintes critérios de paragem:

- Quando se atinge um número máximo de gerações em que a evolução deve ocorrer;
- Quando se atinge um determinado limite temporal (para problemas que exigem soluções em tempo útil ou em tempo real);
- Quando se atinge um valor mínimo no desvio padrão do valor de aptidão dos indivíduos na população;
- Quando não se registam melhorias significativas das soluções ao longo de um dado número de gerações.

2.6 Exemplos de aplicação

Nesta secção iremos apresentar dois exemplos com o objectivo de observarmos melhor algumas características dos AG. Os exemplos usados foram adaptados de [Costa e Simões, 2004], [Ramos, 2001], [Ferrolho e Crisóstomo, 2004d] e [Ferrolho e Crisóstomo, 2005d].

2.6.1 Exemplo do máximo de uma função

Para resumir todos os aspectos referidos até agora, vamos apresentar um exemplo simples de optimização de uma função matemática e ver como evolui o AG ao longo de um conjunto de gerações.

Pretende-se encontrar a valor máximo da função matemática $f(x_1, x_2)$ definida por:

$$f(x_1, x_2) = 21{,}5 + x_1 . sen(4\Pi x_1 + x_2 . sen(20\Pi x_2) \tag{2.4}$$

com $-3{,}0 \le x_1 \le 12{,}1$ e $4{,}1 \le x_2 \le 5{,}8$.

A figura 2.9 apresenta a representação gráfica da referida função.

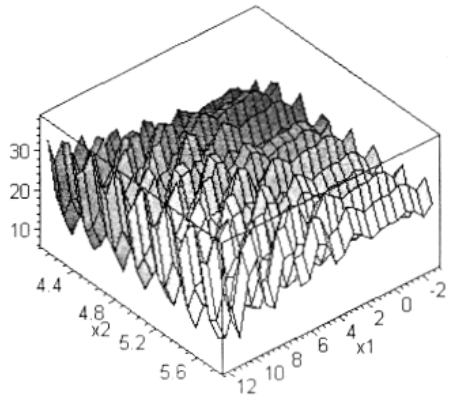


Figura 2.9 Representação gráfica da função.

A função *f* é dependente de duas variáveis (x_1 e x_2). Optando pela representação binária dos cromossomas, a cadeia de *bits* deve codificar a informação referente às duas variáveis.

Para obtermos uma precisão de quatro casas decimais em cada variável, são necessários:

- 18 *bits* para codificar a variável x_1, uma vez que esta possui um domínio de tamanho igual a 15,1 ($2^{17} \le 151000 \le 2^{18}$);
- 15 *bits* para codificar a variável x_2, uma vez que esta possui um domínio de tamanho igual a 1,7 ($2^{14} \le 17000 \le 2^{15}$).

Por conseguinte, são então necessários 33 *bits* para codificar uma solução para a função $f(x_1,x_2)$. Suponhamos o cromossoma representado por: 111101001010111000 001001011000000. Os primeiros 18 *bits* são utilizados para codificar a variável x_1, e os restantes 15 *bits* para codificar a variável x_2.

A equação (2.5) permite encontrar o número real *x* correspondente a um determinado número binário.

$$x = Limite\_Esq\_Dom + x'.\frac{Tamanho\_Dom}{2^{N^\circ\_Bits} - 1} \qquad (2.5)$$

Relativamente à variável x_1, temos a seguinte conversão de binário para decimal:
x'_1=decimal(111101001010111000)=250552.

O número real x_1 correspondente é obtido pela equação (2.5):

$$x_1 = -3{,}0 + 250552 * \frac{15{,}1}{2^{18} - 1} = 11{,}4323 .$$

Utilizando os restantes 15 *bits* do cromossoma referentes à variável x_2, temos:

x'_2=decimal(001001011000000)=4800 e $x_2 = 4{,}1 + 4800 * \frac{1{,}7}{2^{15} - 1} = 4{,}3490 .$

O cromossoma utilizado corresponde ao ponto de coordenadas {11,4323; 4,3490} e o valor dado pela função de avaliação será *f*(11,4323; 4,3490)=13,1733.

Para exemplificar o funcionamento dos operadores de selecção, cruzamento e mutação, vamos simular o funcionamento do AG para uma população de 20 indivíduos. Iremos utilizar o método de selecção por roleta com elitismo em que os dois melhores indivíduos de cada geração irão passar para a geração seguinte. O cruzamento com um ponto de corte e a mutação serão aplicadas com probabilidades de 65% e 1%, respectivamente.

A seguir apresenta-se a população inicial de 20 cromossomas, obtida aleatoriamente:

i_1 = 010011100010111101011110101000111
i_2 = 011011010111111010100010111111000
i_3 = 001100001111100001001111100001010
i_4 = 010111000100111100111100110000110
i_5 = 101111010001110101001101101100101
i_6 = 111011010010111011100001111111010
i_7 = 110000011111000100000101110010011
i_8 = 110001011111110111100011111101111
i_9 = 100010000010101100111011110001101
i_{10} = 111110111111101011010000110110001
i_{11} = 000100101111010101001100111011010
i_{12} = 101001011101011001000010001001000
i_{13} = 011110001110001111110101010011100
i_{14} = 101000101100010111011000011001111
i_{15} = 101010001010010001101100101101000
i_{16} = 001011110100000101100011001100000
i_{17} = 000000100000000110100010111100101
i_{18} = 110110001001001100010101000001010
i_{19} = 011110011100111111011111011000001
i_{20} = 101000001001110100110000000010100

A função utilizada para avaliar os indivíduos é a própria função matemática (2.4). Assim, cada indivíduo terá um valor que indica a sua qualidade (*f(x)*), uma probabilidade de ser seleccionado

(*p(x)*) e é-lhe atribuído um valor (*r(x)*) que representa o valor acumulado da probabilidade de ser seleccionado, como mostra a tabela 2.2. Nesta tabela, o valor da probabilidade de cada indivíduo é calculado dividindo o valor do seu mérito pelo somatório do mérito de todos os indivíduos, sendo este último determinado por $F = \sum_{j=1}^{20} f(i_j) = 434{,}53$.

Tabela 2.2 Valores de *f(x)*, *p(x)* e *r(x)*

Indivíduo *i*	***f(x)***	***p(x)***	***r(x)***
i_1	26,849	0,061	0,061
i_2	24,623	0,056	0,118
i_3	24,750	0,057	0,174
i_4	24,347	0,056	0,230
i_5	25,677	0,059	0,289
i_6	21,028	0,048	0,337
i_7	14,702	0,034	0,370
i_8	26,613	0,061	0,431
i_9	16,521	0,045	0,476
i_{10}	10,491	0,024	0,500
i_{11}	21,428	0,049	0,549
i_{12}	17,122	0,039	0,588
i_{13}	30,268	0,069	0,657
i_{14}	28,323	0,065	0,722
i_{15}	13,478	0,031	0,752
i_{16}	25,725	0,059	0,811
i_{17}	23,548	0,054	0,865
i_{18}	16,200	0,037	0,902
i_{19}	28,801	0,066	0,968
i_{20}	14,036	0,032	1,000

Com base nos resultados da tabela 2.2, obtivemos a roleta apresentada na figura 2.10.

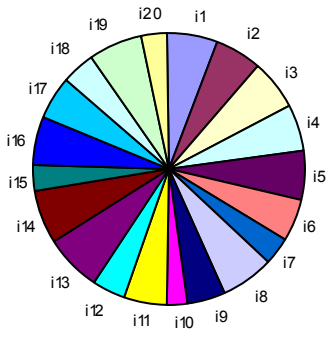


Figura 2.10 Roleta.

Como para a geração seguinte irão transitar os dois indivíduos de melhor qualidade, deveremos seleccionar 9 pares para gerarem 18 novos indivíduos. Os 9 pares foram seleccionados através da

roleta, gerando um número aleatoriamente 18 vezes. Os valores e os indivíduos seleccionados para gerarem descendentes estão indicados na tabela 2.3.

Tabela 2.3 Indivíduos seleccionados

Valores aleatórios [0, 1]	**Indivíduos seleccionados (pares)**
0,79; 0,84	i_{16}, i_{17}
0,84; 0,10	i_{17}, i_2
0,91; 0,24	i_{19}, i_5
0,52; 0,86	i_{11}, i_{17}
0,03; 0,94	i_1, i_{19}
0,67; 0,52	i_{14}, i_{11}
0,47; 0,92	i_9, i_{19}
0,01; 0,39	i_1, i_8
0,64; 0,55	i_{13}, i_{12}

O operador de cruzamento é aplicado com uma probabilidade de 65%, significando que apenas alguns dos pares seleccionados para reprodução serão efectivamente cruzados.

Os valores aleatórios gerados e os respectivos pontos de corte estão na tabela 4. Como se pode observar, valores aleatórios superiores a 0,65 indicam que não ocorre cruzamento.

Tabela 2.4 Pontos de corte, progenitores e filhos

Valores aleatórios [0, 1]	**Ponto de corte**	**Progenitores**	**Filhos**
0,18	23	i_{16}, i_{17}	i'_3, i'_4
0,76	---	i_{17}, i_2	i'_5, i'_6
0,34	24	i_{19}, i_5	i'_7, i'_8
0,07	9	i_{11}, i_{17}	i'_9, i'_{10}
0,90	---	i_1, i_{19}	i'_{11}, i'_{12}
0,45	17	i_{14}, i_{11}	i'_{13}, i'_{14}
0,07	16	i_9, i_{19}	i'_{15}, i'_{16}
0,91	---	i_1, i_8	i'_{17}, i'_{18}
0,42	31	i_{13}, i_{12}	i'_{19}, i'_{20}

Os dois primeiros indivíduos da descendência foram seleccionados por elitismo, isto é, são os dois melhores indivíduos da população anterior.

$i'_1 = i_{13} = 01111000111000111111010101001110 0$
$i'_2 = i_{19} = 01111001110011111101111101100000 1$

Os descendentes i'_3 e i'_4 são gerados pelo cruzamento dos indivíduos i_{16} e i_{17}, utilizando o ponto de corte 23.

i_{16} = 00101111010000010110001 1[001100000
i_{17} = 00000010000000011010001 0[111100101
i'_3 = 00101111010000010110001111110 0101
i'_4 = 00000010000000011010001000110 0000

Os progenitores i_{17} e i_2 não sofrem recombinação, sendo os descendentes cópias destes dois indivíduos ($i'_5=i_{17}$ e $i'_6=i_2$).

Continuando a aplicar o operador de cruzamento, obteríamos a seguinte população:

i'_1 = 01111000111000111111010101001 1100
i'_2 = 01111001110011111101111101100 0001
i'_3 = 00101111010000010110001111110 0101
i'_4 = 00000010000000011010001000110 0000
i'_5 = 00000010000000011010001011110 0101
i'_6 = 01101101011111101010001011111 1000
i'_7 = 00010010100000011010001011110 0101
i'_8 = 00000010011101010100110011101 1010
i'_9 = 01001110001011110101111010100 0111
i'_{10} =01111001110011111101111101100 0001
i'_{11} = 10100010110001011100110011101 1010
i'_{12} = 00010010111101010101100001100 1111
i'_{13} = 10001000001010111011111011000 0001
i'_{14} = 01111001110011110011101111000 1101
i'_{15} = 01001110001011110101111010100 0111
i'_{16} = 11000101111111011110001111110 1111
i'_{17} = 01111000111000111111010101001 1100
i'_{18} = 10100101110101100100001000100 1000
i'_{19} = 01111000111000111111010101001 1100
i'_{20} = 00101111010000010110001100110 0000

Suponhamos que as mutações se localizam nos seguintes pontos:

Filho i'_3 – gene 14

Filho i'_6 – gene 16

Filho i'_{12} – gene 7

Daqui resulta uma nova população que será novamente avaliada. Os genes que sofreram mutação encontram-se marcados a negrito.

i'_1 = 01111000111000111111010101001 1100
i'_2 = 01111001110011111101111101100 0001
i'_3 = 00101111010000**0**10110001111110 0101
i'_4 = 00000010000000011010001000110 0000
i'_5 = 00000010000000011010001011110 0101
i'_6 = 01101101011111101**1**0100010111111 1000
i'_7 = 00010010100000011010001011110 0101
i'_8 = 00000010011101010100110011101 1010

i'_9 = 01001110001011110101111010100011 1
i'_{10} =01111001110011111101111011000001
i'_{11} = 10100010110001011100110011101101 0
i'_{12} = 0001001**1**011110101010110000110011 11
i'_{13} = 10001000001010111101111101100000 1
i'_{14} = 01111001110011110011101111000110 1
i'_{15} = 01001110001011110101111010100011 1
i'_{16} = 11000101111111011110001111110111 1
i'_{17} = 01111000111000111111010101001110 0
i'_{18} = 10100101110101100100001000100100 0
i'_{19} = 01111000111000111111010101001110 0
i'_{20} = 00101111010000010110001100110000 0

Neste momento, conclui-se um ciclo do AG. O processo é agora repetido ao longo de várias gerações. Na tabela 2.5 encontram-se os valores do melhor indivíduo nas primeiras 11 gerações e na geração número 100. Como se pode observar nesta tabela, o AG vai evoluindo para valores mais altos, como é desejado.

Tabela 2.5 Valores do melhor indivíduo

Geração	Melhor indivíduo
1	30,2680
2	30,2680
3	33,4468
4	33,4468
5	33,4468
6	33,4468
7	33,8733
8	34,1697
9	34,1697
10	34,1697
11	35,4115
...	…
100	37,1530

2.6.2 Exemplo de sequenciamento de tarefas numa fresadora CNC

Considere-se o sequenciamento de 9 tarefas {1,2,3,4,5,6,7,8,9} numa fresadora CNC, onde as tarefas com o número impar são executadas por uma ferramenta *fi* e as tarefas com o número par são executadas com uma ferramenta *fp*. O objectivo que se pretende alcançar neste exemplo é minimizar o número de trocas de ferramentas na fresadora CNC.

A codificação vai ser feita por uma sequência de 9 dígitos e a função de avaliação procurará minimizar o número de trocas de ferramentas. Os elementos da população com menor número de trocas de ferramentas serão os escolhidos para a geração seguinte.

O operador de cruzamento de ponto único, cruzamento de dois pontos e cruzamento uniforme, apresentados na subsecção 2.5.4, não podem ser utilizados neste exemplo porque originariam cromossomas descendentes não viáveis (com tarefas repetidas ou tarefas ausentes). Assim, teremos que usar o operador de cruzamento com manutenção de ordem (*order crossover*, OX), para os cromossomas descendentes serem soluções válidas para o problema em questão.

A tabela 2.6 apresenta a população inicial formada por apenas 4 cromossomas e o valor da função de avaliação *f(x)*. Como já referimos, neste exemplo pretende-se que a função de avaliação tenha o menor valor possível, para minimizar o número de trocas de ferramentas.

Tabela 2.6 Elementos da população e o valor da função de avaliação

Elemento	Geração 1	f(x)
1	394165287	6
2	821397465	3
3	136974285	4
4	754381296	7

Aplica-se então o operador de cruzamento com manutenção de ordem e 2 pontos de corte (entre o 2.º e 3.º genes e entre o 6.º e 7.º genes). Cruzando o elemento 1 com o 2 e o elemento 3 com o 4, obtemos:

P1 = 39 - 4165 - 287 = 287394165

P2 = 82 - 1397 - 465 = 465821397

P3 = 13 - 6974 - 285 = 285136974

P4 = 75 - 4381 - 296 = 296754381

Primeiro, a parte central do cromossoma P1 (4165) é eliminada no cromossoma P2, resultando a seguinte sequência de tarefas 82397 (P2 = 465821397 = 82397). De seguida, a parte central do

cromossoma P2 (1397) é eliminada no cromossoma P1, dando origem à sequência de tarefas 28465 (P1 = 28739416**5** = 28465).

Os cromossomas descendentes D1 e D2 (elementos 5 e 6 da tabela 2.7) são obtidos da seguinte forma:

D1 = 97 - 4165 - 823 = 974165823

D2 = 65 - 1397 - 284 = 651397284

Aplicando o mesmo processo aos cromossomas progenitores P3 e P4, obtemos os cromossomas descendentes D3 e D4 (elementos 7 e 8 da tabela 2.7).

Tabela 2.7 Geração 1 após a aplicação do operador cruzamento

Elemento	Geração 1	f(x)
1	39-4165-287	6
2	82-1397-465	3
3	13-6974-285	4
4	75-4381-296	7
5-cruz.1,2	97-4165-823	6
6-cruz.1,2	65-1397-284	2
7-cruz.3,4	81-6974-253	5
8-cruz.3,4	97-4381-256	7

A tabela 2.8 mostra a geração 2 obtida por elitismo. Desta forma, é garantida a passagem dos melhores indivíduos para a geração seguinte.

Tabela 2.8 Geração 2

Elemento	Geração 2	f(x)
1	821397465	3
2	136974285	4
3	651397284	2
4	816974253	5

Relativamente aos resultados obtidos na tabela 2.8, podemos efectuar os seguintes comentários:

- Foi possível obter um elemento melhor que o da geração anterior, para além da população ter melhores funções de avaliação, que neste caso interessa minimizar;
- A solução óptima, para este caso, seria aquela em que fossem realizadas todas as tarefas pares e depois todas as tarefas impares ou vice-versa, com custo de 1.

2.7 Resumo do Capítulo

Neste capítulo foram revistos alguns conceitos sobre AG que serão usados na resolução de problemas de sequenciamento (capítulos 5, 6 e 7). Assim, foram apresentados os conceitos básicos da selecção natural, bem como foram descritos os aspectos fundamentais dos AG e, por último, apresentados dois exemplos com o objectivo de melhor observarmos algumas características dos AG.

Um AG trabalha com base na geração e na evolução contínua de populações de cromossomas. Cada estrutura de cromossoma codifica uma solução do problema e representa um ponto no espaço de busca, estando a sua aptidão relacionada com o valor da função objectivo. Os AG constituem um instrumento novo de pesquisa, capazes de serem utilizados em diversas áreas das actividades humanas. A sua principal vantagem reside na diversidade genética, ou seja, a cada passo do processo de busca, um conjunto de candidatos é considerado e envolvido na criação de novos candidatos.

O sucesso de um AG está dependente do método de codificação das soluções candidatas, dos tipos de operadores genéticos utilizados, dos ajustes dos parâmetros, entre outros. Um AG na sua forma básica pode ser insuficiente para um determinado tipo de problema, por exemplo: a codificação *bit-string* pode ser inadequada; a selecção através da proporcionalidade do *fitness* pode conduzir à convergência prematura; os operadores de cruzamento e de mutação podem tornar inviáveis as soluções descendentes, bem como podem destruir bons esquemas de cromossomas actuais; um ajuste inadequado dos parâmetros pode diminuir a performance global do algoritmo. Por este motivo, no desenvolvimento de um AG, as decisões sobre cada passo devem ser analisadas cuidadosamente, visto que existem muitas possibilidades de implementação e o relacionamento entre os objectos de decisão geralmente não estão suficientemente claros e simples.

Capítulo 3

Robôs Manipuladores e Máquinas CNC Industriais: controlo e software distribuído

"Inventing is mixing brains and materials.
The more brains you use, the less materials you need."
Charles F. Kettering

3.1 Introdução

No âmbito deste capítulo, foram desenvolvidas várias ferramentas de software para utilização em robôs e máquinas CNC industriais. O grande objectivo destas ferramentas de software é permitir a integração destes equipamentos, de uma maneira fácil e eficiente, nos modernos sistemas de produção. Para os robôs manipuladores industriais foram desenvolvidas as ferramentas de software "winRS232ROBOTcontrol" e "winEthernetROBOTcontrol". Para as máquinas CNC industriais, foi desenvolvida a ferramenta de software "winMILLcontrol" especificamente para as fresadoras e "winTURNcontrol" para os tornos. As ferramentas de software desenvolvidas utilizam as potencialidades originais dos sistemas de controlo dos robôs e das máquinas CNC, num ambiente distribuído cliente/servidor. Com estas ferramentas, os robôs e as máquinas CNC industriais podem ser integrados, de forma fácil e eficiente, nos mais modernos sistemas de produção.

A arquitectura apresentada neste capítulo é aplicável a equipamentos com as seguintes características: capacidade de processamento local (microprocessador), possibilidade de armazenamento de informação e capacidade de comunicação em rede usando protocolos TCP/IP ou, na ausência deste, possibilidade de comunicação RS232. A grande maioria dos equipamentos modernos possui estas características, pelo que a aplicabilidade da abordagem aqui apresentada não está comprometida. Os programas de gestão, controlo, integração e coordenação dos diversos equipamentos podem ser desenvolvidos usando qualquer linguagem de programação *standard* (*C++*, *C++ Builder*, *Visual Basic*). O assunto não será esgotado neste trabalho, pois para além de robôs manipuladores e máquinas CNC industriais devem ser considerados, também, outros equipamentos que podem fazer parte dos modernos sistemas de produção. Para

esses equipamentos deve também ser definida uma biblioteca de funções de forma a ser possível a sua integração e controlo.

As ferramentas de software desenvolvidas no âmbito deste capítulo têm os seguintes objectivos:

1. Controlar e monitorar remotamente todas as funcionalidades dos robôs e máquinas CNC industriais. O controlo e monitorização remota são determinantes para a integração dos equipamentos nos modernos sistemas de produção.
2. Desenvolver arquitecturas de software distribuído para utilização em redes *Ethernet*, baseadas em computadores pessoais.
3. Permitir o desenvolvimento de aplicações que possibilitem a exploração remota dos robôs manipuladores e máquinas CNC usando sempre a mesma linguagem de programação (*e.g.*, *C++*).
4. Desenvolvimento de interfaces gráficas (*graphical user interfaces* – GUI) para robôs e máquinas CNC industriais, de forma simples e rápida.
5. Desenvolvimento de software de controlo para robôs e máquinas CNC de forma mais fácil, barata e flexível. Como resultado, os engenheiros das empresas proprietárias dos sistemas de produção poderão efectuar, sem terem que recorrer a terceiros, as alterações que entenderem aos programas de controlo dos equipamentos.
6. Permitir o aproveitamento das potencialidades avançadas dos sistemas de controlo que equipam os robôs industriais, nomeadamente a capacidade de programação local usando uma linguagem dedicada com extensas bibliotecas de funções, as capacidades de manipulação, o software de planeamento de trajectórias que recorre aos modelos cinemático e dinâmico optimizados da estrutura mecânica, etc..
7. Permitir a utilização de uma rede de robôs e máquinas CNC em ambiente industrial, permitindo a implementação de software de gestão, coordenação, integração e controlo em processos de produção, como é o caso das células flexíveis de fabrico, segundo uma estrutura hierárquica do tipo CIM.
8. Integrar e controlar vários equipamentos de diferentes fabricantes em sistemas de produção, como por exemplo: transportadores, motores, sensores, entre outros (ver o software e hardware USB desenvolvido na secção 3.6).

Tendo por base estes objectivos, propõe-se neste capítulo uma arquitectura de software que utiliza as capacidades do sistema de controlo originais dos robôs e das máquinas CNC, num ambiente distribuído cliente/servidor. As ferramentas de software desenvolvidas e apresentadas

neste capítulo foram utilizados no desenvolvimento de uma Célula Flexível de Fabrico (CFF), apresentada no próximo capítulo.

Este capítulo está estruturado da seguinte forma: na secção 3.2 encontra-se um resumo dos trabalhos mais relevantes encontrados na literatura; na secção 3.3 é apresentada genericamente a arquitectura utilizada no controlo e monitorização remota de robôs manipuladores e máquinas CNC industriais; a secção 3.4 mostra as ferramentas de software desenvolvidas para utilização em robôs industriais; a secção 3.5 aborda as várias ferramentas de software desenvolvidas para as máquinas CNC industriais; a secção 3.6 apresenta o software e hardware USB desenvolvido com a finalidade de permitir a integração e o controlo de transportadores, motores, sensores, actuadores, entre outros, em SFF e CFF e, por último, na secção 3.7 encontra-se um resumo do trabalho desenvolvido ao longo deste capítulo.

3.2 Revisão da Literatura

O aparecimento dos Sistemas Flexíveis de Fabrico (SFF), conhecidos na literatura inglesa como *Flexible Manufacturing Systems* (FMS), proporcionou um impacto positivo no melhoramento da indústria de manufactura mundial. Os SFF podem ser definidos de muitas formas, mas não existe uma definição *standard* unanimemente aceite pela comunidade científica. Muitas das definições são baseadas no hardware usado no sistema. Por exemplo, Byrkett em [Byrkett *et al.*, 1988] diz o seguinte:

> *"A flexible manufacturing system (FMS) is a manufacturing system in which groups of numerical controlled machines (machine centers) and a material handling system work together under computer control."*

Keefe e Kasirajan em [Keefe e Kasirajan, 1992] definem SFF como:

> *"A group of workstations connected together by a material handling system (MHS) producing or assembling a number of different part types under the central control of a computer."*

Outras definições são baseadas na capacidade ou performance do sistema. Por exemplo, Kaltwasser em [Kaltwasser *et al.*, 1986] define SFF da seguinte forma:

> *"Flexible Manufacturing Systems (FMS) are highly automated production systems, able to produce a great variety of different parts by using the same equipment and the same control system."*

Outras definições podem ser encontradas na literatura [Chan *et al.*, 2002], [Sarin e Chen, 1987] e [Grady e Menon, 1986]. Das várias definições apresentadas na literatura, podemos concluir que um SFF é constituído basicamente por três subsistemas:

1. Sistema de processamento.
2. Sistema de transporte e armazenamento.
3. Sistema de controlo por computador.

Uma CFF, na literatura inglesa *Flexible Manufacturing Cell* (FMC), pode ser definida como uma "configuração integrada e controlada por computador de máquinas-ferramenta de controlo numérico, equipamento auxiliar de produção e um sistema de manuseamento de materiais, concebida para produzir simultaneamente pequenas a médias séries de uma vasta gama de produtos, de elevada qualidade, a baixo custo" [Viswanadham e Narahari, 1992]. Da definição verificamos que uma CFF pode incluir robôs manipuladores, AGV (*Automated Guided Vehicles*), autómatos programáveis, sistemas de visão, máquinas CNC, transportadores, *buffers*, armazéns, entre outros. Estes equipamentos, mesmo sendo do mesmo fabricante, têm linguagens de programação diferentes. Torna-se assim complicado e demorado alterar a funcionalidade da célula para adaptá-la às exigências de um novo produto ou de um produto que sofreu alterações. Muitos trabalhos têm sido publicados sobre este assunto, propondo soluções que de uma maneira geral apontam para o desenvolvimento de uma linguagem de programação única e de interpretadores para as linguagens dos vários equipamentos. Trabalhos recentes mostram que é possível e desejável ter um ambiente flexível e continuar a programar cada equipamento usando a sua linguagem original [Magy e Haidegger, 1993] e [Li, 1994].

Os SFF ou as CFF são constituídos por duas grandes partes: hardware e software de controlo. Do hardware podem fazer parte máquinas CNC, robôs, armazéns, sistemas de transporte, entre outros. Os problemas associados ao hardware têm vindo a ser estudados nas últimas décadas e actualmente são razoavelmente bem entendidos [Joshi *et al.*, 1995a]. No entanto, desenvolver software de controlo para um SFF ou uma CFF não é uma tarefa fácil [Joshi *et al.*, 1995a], [Joshi e Mettala, 1991], [Joshi *et al.*, 1995b] e [http#1]. Actualmente, o software é desenvolvido de acordo com a especificidade do sistema, tornando-se muito dispendioso e difícil de alterar. Muitos SFF e CFF que se encontram a funcionar actualmente nas empresas foram desenvolvidos e vendidos por empresas de integração de sistemas. Como resultado, o conhecimento relativo ao software de controlo não fica na posse das empresas que adquirem os SFF ou as CFF, ficando

estas totalmente dependentes das empresas de integração de sistemas, sempre que pretendam efectuar alterações ou melhorias no software.

Joshi, já em 1995, apresentou alguns modelos de controlo com o objectivo destes poderem ser utilizados no controlo de CFF [Joshi *et al.*, 1995a]. Outros autores desenvolveram protótipos de sistemas para desenvolvimento rápido de software de controlo [Maimon e Fisher, 1988]. Hadj--Alouane em [Hadj-Alouane *et al.*, 1988] apresenta uma descrição detalhada sobre componentes de hardware e software em sistemas de transporte de material, com o objectivo destes serem integrados em sistemas de produção.

Investigadores da *Pennsylvania State University's Computer Integrated Manufacturing Laboratory* (CIMLAB) têm vindo a trabalhar, há mais de uma década, na simulação do controlo de sistemas de produção, tendo desenvolvido uma ferramenta de trabalho chamada "RapidCIM". O "RapidCIM" foi inicialmente desenvolvido para o controlo de sistemas de produção discreta, tendo sido implementada numa plataforma de controlo de tempo real. Através das ferramentas disponíveis no "RapidCIM" é possível gerar automaticamente muito do software necessário para os sistemas de produção discretos. Assim, é possível reduzir de forma significativa os custos de desenvolvimento e integração nestes sistemas. Mais informação sobre o "RapidCIM" pode ser encontrada em [Joshi *et al.*, 1995b], [http#1], [http#2] e [http#3].

3.3 Arquitectura Proposta

Para cada equipamento (*e.g.*, robôs manipuladores, máquinas CNC, entre outros) foi desenvolvida uma biblioteca de funções com a finalidade de permitir a coordenação, integração, controlo e exploração remota desses equipamentos em SFF e CFF. Sempre que necessário, deverá ser fácil adicionar novas funções à biblioteca inicial de cada equipamento. No entanto, a biblioteca de cada equipamento deverá ser o mais completa possível para que em condições de funcionamento normal se torne desnecessário adicionar novas funções. De facto, quando integrado numa CFF qualquer equipamento tem um conjunto bem definido de tarefas que deve ser capaz de executar. Essas tarefas devem ser identificadas, e as funções que as realizam devem ser implementadas de uma forma o mais geral possível, com o objectivo de poderem ser usadas em diferentes aplicações ou até em diferentes CFF.

Neste trabalho foram utilizados os seguintes modelos de programação:

1. Cliente/Servidor – Código servidor a correr em cada equipamento, que recebe chamadas de computadores remotos (clientes), as executa, e devolve os respectivos resultados.
2. Chamadas Remotas (RPC - *Remote Procedure Calls*) – Este é o método mais usual para implementar comunicações entre um cliente e o servidor de aplicações distribuídas. O cliente faz uma chamada e o mecanismo RPC transforma-a numa chamada via rede, adicionando o necessário para estabelecer a comunicação em rede. O servidor recebe o pedido do cliente e executa-o de acordo com a parametrização fornecida, devolvendo os resultados.
3. Partilha de Dados (*Data Sharing*) – Pretende-se que existam serviços de partilha de ficheiros, programas, bases de dados, entre outros, entre o(s) cliente(s) e o(s) servidor(es).

Em suma, a arquitectura genérica proposta funciona num ambiente de programação cliente/servidor. Esta baseia-se na utilização de serviços remotos através da realização de chamadas remotas entre o computador e o controlador do equipamento em causa.

3.4 Software para robôs manipuladores industriais

Um robô industrial é controlado por um controlador (sistema electrónico computorizado) que controla toda a estrutura mecânica, coordena a acção dos motores das articulações (de acordo com informação sensorial de posição e velocidade) e o movimento a executar pela estrutura mecânica. Isso significa capacidade de programação local, armazenamento de informação, bem como, algoritmos de controlo e planeamento de trajectórias que implicam a existência de modelos cinemáticos e dinâmicos que descrevam a estrutura mecânica e o respectivo movimento. Dadas as exigências de precisão e velocidade das actuais estruturas de produção, aliadas a uma constante evolução de produtos que exige dos equipamentos de produção uma grande flexibilidade, os sistemas de controlo dos robôs industriais são actualmente sistemas muito avançados. Estes sistemas disponibilizam uma linguagem de programação estruturada (semelhante ao PASCAL) [Nilsson, 1992] [Nilsson, 1996], bibliotecas de funções avançadas de manipulação e controlo de posição, possibilidade de gestão de programas e ficheiros (na maioria deles), bem como dispositivos de interface com o utilizador (geralmente um *teach-pendant* com LCD, um teclado, botão de paragem de emergência e, eventualmente, um *joystick*).

Para muitas aplicações, nomeadamente para aquelas que não exijam coordenação com outros equipamentos, os robôs industriais actuais estão razoavelmente adaptados. Mesmo para sistemas de produção (pouco exigentes!) não existem grandes problemas em aplicar os actuais robôs industriais, desde que os respectivos programas sejam desenvolvidos especificamente para a aplicação em causa. Os problemas surgem quando pretendemos desenvolver uma CFF com um ou mais robôs, uma ou mais máquinas CNC, juntamente com outros equipamentos, com a finalidade destes trabalharem de forma integrada e coordenada. Sempre que utilizamos equipamentos de fabricantes diferentes numa CFF enfrentamos, na maioria das vezes, problemas de difícil resolução.

Hoje os robôs manipuladores são construídos como sistemas essencialmente locais, apesar de alguns fabricantes disponibilizarem meios para acesso remoto, destinados essencialmente a operações de monitorização de estado e gestão de programas e ficheiros (*e.g.*, *backups*, gestão de programas, etc.). A utilização de redes industriais, interligando os controladores dos robôs e outros equipamentos, permite a construção de aplicações distribuídas baseadas em computadores pessoais, aliando à sua capacidade de processamento e de programação as funcionalidades já evidenciadas dos controladores dos robôs. As vantagens são evidentes:

1. Gestão e monitorização de todos os equipamentos envolvidos na CFF, e por conseguinte, de todo o processo produtivo.
2. Facilidade de programação e teste, visto que se utilizam ferramentas *standard* baseadas em PC.
3. Possibilidade de implementar estruturas de controlo inteligentes, que respondem a solicitações externas devidamente parametrizadas. Qualquer equipamento da célula de produção (um robô/controlador ou uma máquina CNC) disponibilizará um conjunto de funcionalidades que podem ser solicitadas remotamente. O pedido de execução pode ser feito por qualquer computador da rede desde que possua as devidas permissões de acesso, estando por isso reservadas para o(s) computador(es) que gere(m) a CFF.

Tendo em conta o que foi discutido anteriormente, apresentamos algumas ideias tidas em conta no software desenvolvido:

1. O software deve ser distribuído, baseado numa estrutura cliente/servidor. Pretende-se usar ao máximo os sistemas locais de programação, controlo e gestão. Estes devem ser utilizados, tendo em conta que resultam de muitos anos de desenvolvimento e aperfeiçoamento difíceis de substituir. Para além disso, o objectivo principal é a

possibilidade dos sistemas individuais (robôs/controladores, máquinas CNC, etc.) disponibilizarem as suas funcionalidades sob a forma de serviços remotos.

2. Deve ainda permitir a integração em sistemas de produção organizados segundo uma estrutura hierárquica do tipo CIM. Na verdade, deve estar vocacionado para esse tipo de estrutura de integração e controlo.
3. Deve permitir, de forma simples, o desenvolvimento de interfaces com o utilizador que disponibilizem informação *on-line* sobre os robôs, máquinas CNC e outros equipamentos usados numa determinada CFF.
4. Deve funcionar em plataformas *standard* largamente divulgadas em ambiente industrial.
5. Deve possibilitar a integração fácil de novas funcionalidades usando os ambientes de programação largamente divulgados.

Em suma, os robôs manipuladores que se encontram a funcionar actualmente nas mais variadas empresas têm algumas limitações, nomeadamente:

1. Alguns controladores de robôs permitem comunicações do tipo RS232, enquanto outros permitem o uso de redes *Ethernet*.
2. Os robôs manipuladores possuem controladores fechados. Estes dispositivos de controlo de posição são capazes de receber uma trajectória para o manipulador executar e correrem-na continuamente.
3. Muitos controladores de robôs manipuladores não permitem o controlo remoto a partir de computadores externos.
4. O ambiente de programação dos robôs manipuladores não é suficientemente poderoso para permitir o desenvolvimento de programas que necessitem de técnicas de controlo complexas (*e.g.*, integração e controlo de robôs em CFF).
5. Robôs manipuladores de diferentes fabricantes têm linguagens e ambientes de programação diferentes, isto é, cada fabricante tem a sua linguagem e ambiente de programação. Assim, a integração e coordenação de robôs de diferentes fabricantes é uma tarefa difícil.

Com a finalidade de reduzirmos muitas destas limitações, desenvolvemos as ferramentas "winRS232ROBOTcontrol" e "winEthernetROBOTcontrol" [Ferrolho e Crisóstomo, 2007a]. Com as ferramentas de software desenvolvidas é possível:

1. Desenvolver arquitecturas distribuídas de software, baseadas em computadores pessoais, usando redes *Ethernet*.

2. Desenvolver aplicações que permitam a exploração remota de robôs manipuladores usando sempre a mesma linguagem de programação (*e.g.*, C++).
3. Carregar programas nos controladores dos robôs a partir de qualquer computador de uma rede *Ethernet*.
4. Efectuar a gestão de programas e ficheiros, incluindo ficheiros de registo de operações.
5. Proceder à recuperação de situações de paragem de emergência, colocando os robôs em posições de segurança. Estes procedimentos utilizam trajectórias de segurança que têm em conta a posição actual e estado do robô, bem como, da ferramenta em uso.
6. Usar computadores pessoais como meio de programação, tirando partido de um grande número de ferramentas de programação e análise disponíveis nos mesmos.
7. Desenvolver, de forma simples e rápida, GUI para robôs manipuladores.

3.4.1 *winRS232ROBOTcontrol*

O "winRS232ROBOTcontrol" foi desenvolvido com o intuito deste poder ser utilizado em robôs manipuladores com controladores que permitam uma comunicação série RS232. Com o "winRS232ROBOTcontrol" é possível desenvolver programas para os robôs manipuladores em C++, sendo estes executados ao nível do computador e não no controlador do robô. No entanto, o "winRS232ROBOTcontrol" também permite o desenvolvimento de programas para serem executados ao nível do controlador. Para além destas duas soluções, também é possível ter programas a correr no controlador e no computador (solução mista), em simultâneo ou individualmente. O "winRS232ROBOTcontrol" foi desenvolvido para robôs manipuladores industriais e utilizado no robô *Scorbot ER VII*. No entanto, o "winRS232ROBOTcontrol" foi pensado para ser utilizado por todos os robôs que suportem comunicações RS232. Evidentemente, sempre que se pretender utilizar o "winRS232ROBOTcontrol" noutro tipo de robô manipulador é necessário efectuar actualizações na biblioteca *RobotControlAPI* (API - *Application Programming Interface*), de acordo com o robô a ser utilizado [Ferrolho e Crisóstomo, 2003] e [Ferrolho e Crisóstomo, 2007a].

A interface gráfica do "winRS232ROBOTcontrol" foi desenvolvida em *Borland C++ Builder* [Alves, 2002]. Esta interface disponibiliza uma extensa biblioteca de componentes reutilizáveis e um ambiente de ferramentas RAD (*Rapid Application Development*). O desenvolvimento da parte de baixo nível da biblioteca *RobotControlAPI* foi realizado no *Visual C++* [Kruglinski, 1997].

Para o "winRS232ROBOTcontrol" foi desenvolvido um processo paralelo (*thread*) com o objectivo de fazer o *polling* do porto série. Sempre que aparecem dados no porto série é gerado um evento e o envio de dados para o controlador do robô é realizado através do envio de mensagens para a fila de espera (*queue*) do porto série. Por exemplo, para abrir o *gripper* do robô é colocada uma mensagem na fila de espera do processo paralelo e assim que a mensagem é recebida são retirados os dados da fila de espera e enviados para o controlador através do canal RS232. Após o envio dos dados é activado um ciclo de espera de resposta do controlador, ciclo este que terminará quando o controlador enviar a *prompt* (>), quando acontecer um *timeout* ou ocorrer uma mensagem de cancelamento. Na figura 3.1 é possível visualizar um esquema elucidativo do processo paralelo.

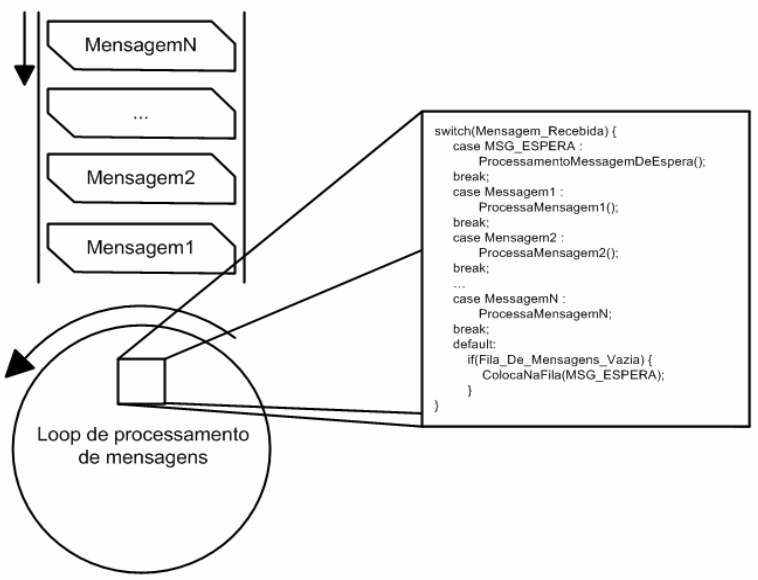


Figura 3.1 Ciclo de mensagens no processo paralelo.

O programa principal comunica com o processo paralelo através das mensagens colocadas na fila de espera, sendo esta fila gerida pelo sistema operativo. Sempre que chega uma mensagem ao processo paralelo esta é de imediato processada. Caso não haja mensagens na fila de espera o processo paralelo entra num *loop*, ficando à "escuta" dos dados vindos do controlador e sempre que ocorram é gerado um evento.

A figura 3.2 ilustra a estrutura hierárquica das principais classes que compõem a biblioteca *RobotControlAPI* desenvolvida, sendo a classe *RobotControl::BasicThread* a de mais baixo nível e a classe *RobotControl::Controller* a de mais alto nível.

A classe *RobotControl::BasicThread* é responsável por lidar com a gestão do processo paralelo. A classe *RobotControl::Terminal* herda da classe base *RobotControl::BasicThread* todos os métodos que lhe permitem controlar processos paralelos. Esta classe funciona como um emulador de terminal e permite a captura de texto no canal RS232. A classe *RobotControl::Controller* é a classe de mais alto nível da API e herda todas as funcionalidades da classe *RobotControl::Terminal*. Uma das suas funções é o envio e a recepção de dados para posterior análise.

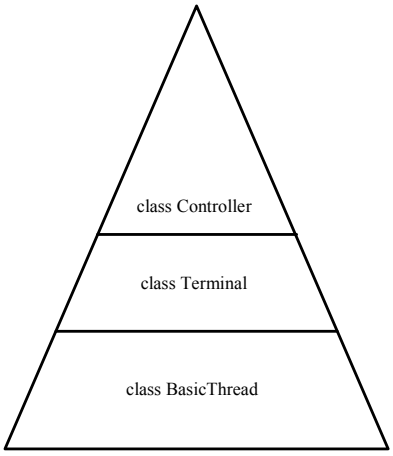


Figura 3.2 Estrutura hierárquica da API.

A biblioteca *RobotControlAPI* desenvolvida para a aplicação "winRS232ROBOTcontrol" é baseada num processo paralelo que corre em simultâneo com o fluxo do programa principal. Esta biblioteca foi desenvolvida com o objectivo de aceder às funções do controlador, como ilustra a figura 3.3. A comunicação com o controlador do robô só é possível graças à biblioteca *RobotControlAPI*, estando as suas funções divididas em dois grupos. No primeiro grupo estão os métodos públicos e no segundo grupo os métodos que representam eventos. A tabela 3.1 ilustra algumas destas funções.

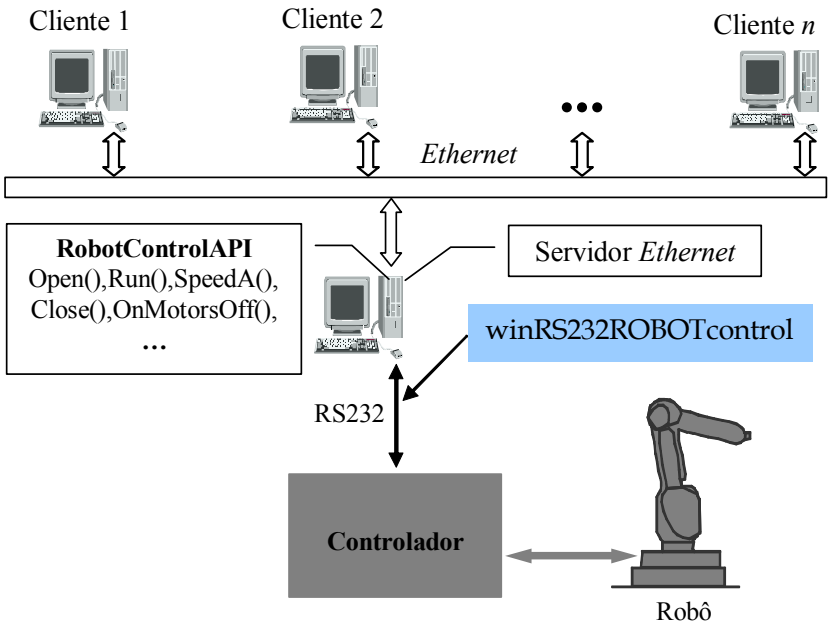


Figura 3.3 Biblioteca *RobotControlAPI* e *winRS232ROBOTcontrol*.

Tabela 3.1 Algumas funções disponíveis

	Função	Descrição		Função	Descrição
Métodos públicos	SpeedA()	Altera a velocidade do robô	**Eventos**	OnEndHoming()	Este evento é activado no fim do *homing*
	Home()	Faz o *homing* ao robô		OnClose()	Este evento fica activo quando se executa a função Close()
	MotorsOff()	Desliga os motores do robô		OnOpen()	Este evento fica activo quando se usa a função Open()
	MotorsOn()	Liga os motores do robô		OnMotorsOff()	Este evento fica activo sempre que os motores são desligados
	Close()	Fecha o *gripper* do robô		OnMotorsOn()	Este evento fica activo sempre que os motores são ligados
	Open()	Abre o *gripper* do robô			

Foi desenvolvido um servidor *Ethernet* para ser utilizado pelo computador que controla o robô manipulador. Desta forma, e como mostra a figura 3.3, é possível controlar o robô remotamente, num ambiente distribuído cliente/servidor.

3.4.2 *winEthernetROBOTcontrol*

O "winEthernetROBOTcontrol" foi desenvolvido para ser utilizado em robôs manipuladores industriais com controladores que permitam uma ligação e comunicação *Ethernet*. A biblioteca de funções disponibilizada em "winEthernetROBOTcontrol" permite desenvolver aplicações, de forma simples e rápida, para robôs industriais. O "winEthernetROBOTcontrol" utiliza as potencialidades originais dos controladores dos robôs, num ambiente distribuído cliente/servidor. A figura 3.4 mostra os computadores PC1 e PC4 a comunicarem com os robôs através do "winEthernetROBOTcontrol" [Ferrolho e Crisóstomo, 2007a].

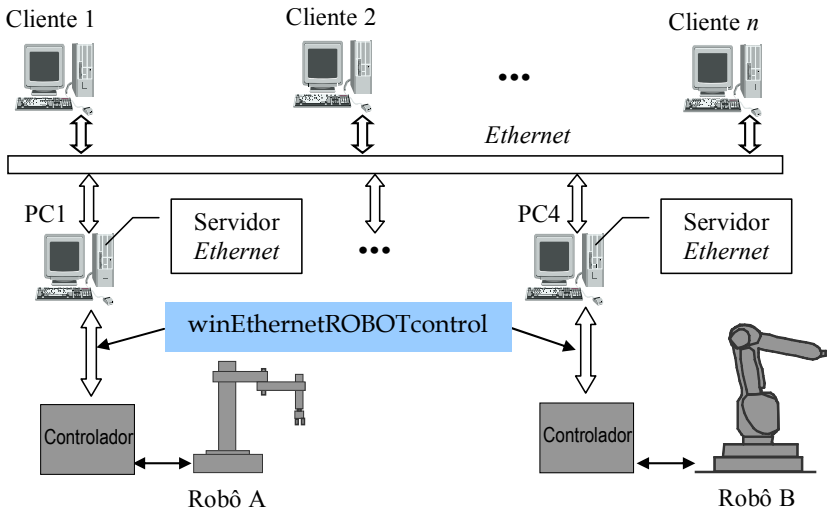


Figura 3.4 *winEthernetROBOTcontrol*.

A tabela 3.2 apresenta alguns dos procedimentos, funções e eventos desenvolvidos no âmbito do "winEthernetROBOTcontrol". Por exemplo, o procedimento "Robot" permite efectuar uma comunicação entre um PC (com "winEthernetROBOTcontrol") e um ou mais robôs. Para isso, é necessário estabelecer e configurar uma ligação entre o PC e o controlador do robô (*Add New Alias*), como mostra a figura 3.5. Depois disso, é possível comunicar e controlar o robô remotamente através da ligação estabelecida (Alias).

Tabela 3.2 Alguns procedimentos, funções e eventos

Procedimentos	**Descrição**
Robot()	*Add New Alias*
S4ProgamLoad()	Lê um programa no controlador do robô.
S4Run()	Liga os motores do robô.
S4Standby()	Desliga os motores do robô.
S4Start()	Inicia a execução de um programa.
S4Stop()	Pára a execução de um programa.
S4ProgramNumVarRead()	Lê uma variável numérica no controlador do robô.
S4ProgramNumVarWrite()	Escreve uma variável numérica no controlador do robô.
S4ProgramStringVarWrite()	Escreve uma *string* no controlador do robô.
S4ProgramStringVarRead()	Lê uma *string* no controlador do robô.

Funções	**Descrição**
Add_Variable()	Adiciona uma nova variável.
Remove_Variable()	Remove uma variável.
ControlID()	Devolve o ID do controlador.
InerfaceState()	Devolve o estado da interface.

Eventos	**Descrição**
StatusChanged()	Este evento fica activo sempre que algo muda de estado no controlador do robô. Por exemplo, desligar motores, ligar motores, emergência activa, correr um programa, entre outros.
BoolVariableChanged()	Este evento fica activo sempre que uma variável *boolean* muda de estado.
NumVariableChanged()	Este evento fica activo sempre que uma variável numérica muda.

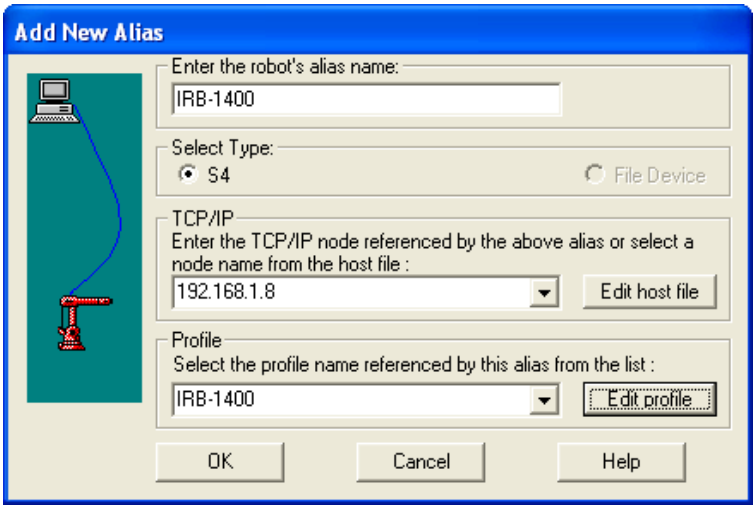


Figura 3.5 *Add New Alias.*

A biblioteca desenvolvida em "winEthernetROBOTcontrol" pode ser utilizada em muitos dos robôs manipuladores actualmente usados na indústria. Como veremos na subsecção 3.4.4 e no próximo capítulo, esta biblioteca foi utilizada com sucesso no desenvolvimento de aplicações para robôs industriais da marca ABB, com o novo sistema de controlo *S4Cplus*. No entanto, para que o "winEthernetROBOTcontrol" possa ser utilizado num robô de outra marca é necessário fazer algumas actualizações na biblioteca, de acordo com as características do robô em questão. A ideia base é definir uma biblioteca de funções para cada robô de forma a permitir o controlo remoto dos mesmos, num ambiente distribuído cliente/servidor.

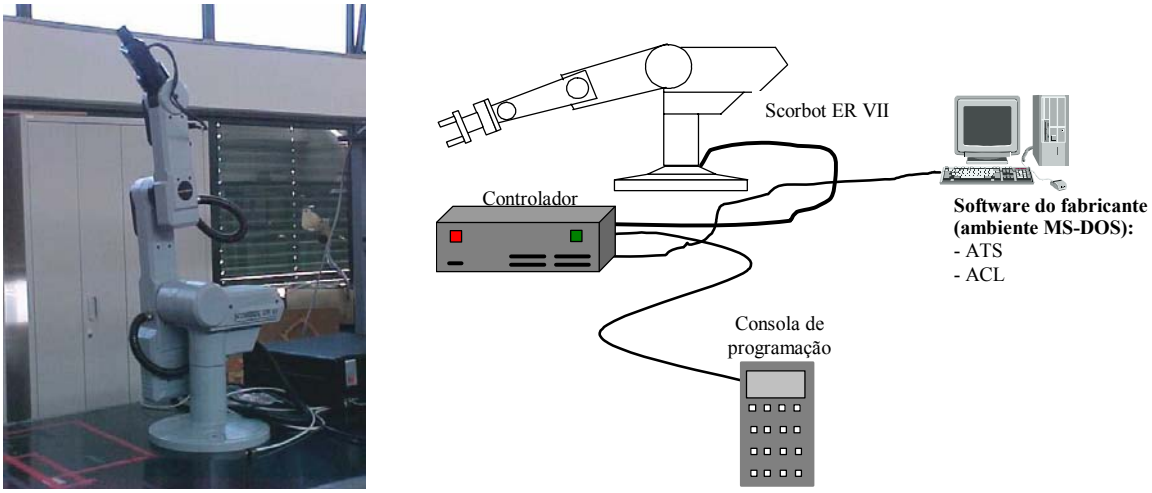
3.4.3 Exemplo de aplicação do *winRS232ROBOTcontrol*

De forma a testarmos as potencialidades do "winRS232ROBOTcontrol", este foi utilizado com sucesso no desenvolvimento de ferramentas de controlo e monitorização para o robô industrial *Scorbot ER VII*, fabricado pela *Eshed Robotec*. O robô *Scorbot ER VII*, ilustrado na figura 3.6(a), é um robô manipulador de média dimensão, estando especialmente vocacionado para operações de manipulação. Este robô está equipado com um *gripper* eléctrico, sendo possível a sua substituição por outro tipo de ferramenta, ficando o robô apto a realizar outro tipo de tarefas, como por exemplo: pintura, polimento de superfícies ou operações de soldadura. Este robô possui cinco articulações de rotação, sendo desta forma um robô com cinco graus de liberdade [Eshed Robotec, 1995a] e [Eshed Robotec, 1996].

O robô *Scorbot ER VII* é constituído pelo manipulador, controlador, consola de programação e pelas ferramentas de software ATS (*Advanced Terminal Software*) e ACL (*Advanced Control Language*), como mostra a figura 3.6(b). Tanto o ATS como o ACL funcionam em ambiente MS-DOS [Eshed Robotec, 1995a], [Eshed Robotec, 1995b]. O ATS permite o funcionamento *on-line* com o robô, através duma comunicação série RS232. É possível, por exemplo, fazer um *open* e *close* do *gripper* do robô, fazer o *homing* do robô, visualizar os programas residentes em memória e solicitar a execução dos mesmos, efectuar a configuração do robô, entre outros [Eshed Robotec, 1995a].

O ACL funciona em modo *off-line* permitindo o desenvolvimento de programas e o *download* dos mesmos para o controlador do robô, através da porta série RS232 [Eshed Robotec, 1995b] e [Eshed Robotec, 1995c]. Para que o robô possa executar um determinado programa é necessário efectuar sempre o *download* do mesmo para o controlador, uma vez que o programa corre no controlador e não no PC. Ambas as ferramentas de software disponibilizadas pelo fabricante

apresentam limitações ao nível do controlo e monitorização do robô e, também, ao nível da sua programação, para além destas ferramentas correrem em ambiente MS-DOS.



a) b)

Figura 3.6 Robô *Scorbot ER VII.*

O controlador do *Scorbot ER VII* possui um CPU *Motorola* 68020 e 128 Kbytes de RAM, sendo desta forma impossível o desenvolvimento de grandes e exigentes aplicações para o robô. Com a finalidade de ultrapassar estas limitações, foi utilizado o "winRS232ROBOTcontrol" no desenvolvimento de ferramentas de software para o robô. As ferramentas de software desenvolvidas para o *Scorbot ER VII* correm em ambiente *windows* (95, 98, NT, 2000 e XP) e são amigáveis do ponto de vista do utilizador. Com as ferramentas desenvolvidas deixa de haver a obrigatoriedade dos programas estarem na memória do controlador para poderem ser executados pelo robô. Estas ferramentas permitem o desenvolvimento de programas em C++ para o robô, sendo estes executados ao nível do computador e não no controlador do robô. No entanto, também é possível o desenvolvimento de programas que podem ser executados ao nível do controlador do robô. Para além destas duas soluções, também permite uma solução mista, sendo possível ter programas a correr no controlador e no computador, em simultâneo ou individualmente. Com as ferramentas de software desenvolvidas ultrapassaram-se as limitações impostas pelo controlador do robô, uma vez que podemos desenvolver programas para o robô em C++ e correr esses mesmos programas ao nível do computador. A reduzida capacidade de processamento e memória do controlador deixaram de ser impeditivos para o desenvolvimento de grandes e exigentes aplicações para o robô *Scorbot ER VII*.

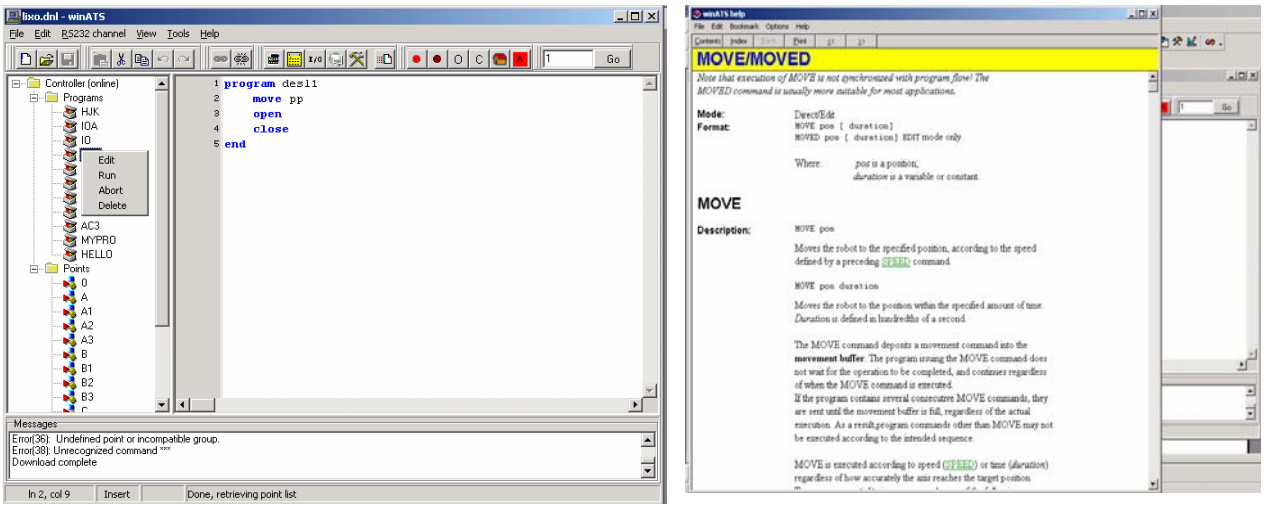
A comunicação faz-se a partir duma porta série do computador e a porta série principal do controlador do robô *Scorbot ER VII*. A comunicação é feita através da transmissão de

informação alfanumérica ASCII (*American Standard Code for Information Interchange*). O código ASCII é um código de sete *bits* aos quais é comum associar um oitavo *bit*, *bit* de paridade, detector de erro simples. Para se estabelecer uma comunicação bidireccional entre o computador e o controlador é necessário configurar os parâmetros do porto série de acordo com a tabela 3.3 [Ferrolho e Crisóstomo, 2003] e [Ferrolho e Crisóstomo, 2004a].

Tabela 3.3 Parâmetros de configuração

Parâmetro	Valor
Taxa de transmissão (*baudrate*)	9600 bps
Bits de paragem	1
Handshake	Software (Xon/Xoff)
Paridade	Nenhuma
Porto	COM1/COM2
Bits de dados	8

A figura 3.7(a) apresenta o editor de programas desenvolvido para o robô *Scorbot ER VII*. Com os ícones disponibilizados pelo editor de programas é possível ligar e desligar os motores do robô, abrir e fechar o *gripper*, fazer o *homing* ao robô, criar e abrir programas, entre outros. No lado esquerdo da figura 3.7(a) é possível ver os programas e os pontos que se encontram na memória do controlador do robô. O editor de programas automaticamente reconhece as palavras reservadas da linguagem ACL e coloca-as com uma cor diferente (azul). Se durante a fase de programação for necessária ajuda relativamente a alguma instrução, basta clicar duas vezes sobre a instrução e automaticamente abre-se uma nova janela com a ajuda pretendida (ver figura 3.7(b)).



a) b)

Figura 3.7 Editor de programas e ficheiro de ajuda (*help*) desenvolvidos.

Na figura 3.8 encontra-se o *teach pendant* desenvolvido com múltiplas funções, tais como: fazer o *homing*, abortar programas, mover o robô, ligar e desligar motores, gravar posições, entre outras. É possível visualizar, com a ajuda da janela de entradas e saídas apresentada na figura 3.9, o estado das entradas e saídas do controlador. Com a janela do *Task Manager* apresentada na figura 3.10 é possível correr, abortar, apagar, suspender e recomeçar um programa ao nível do controlador do robô.

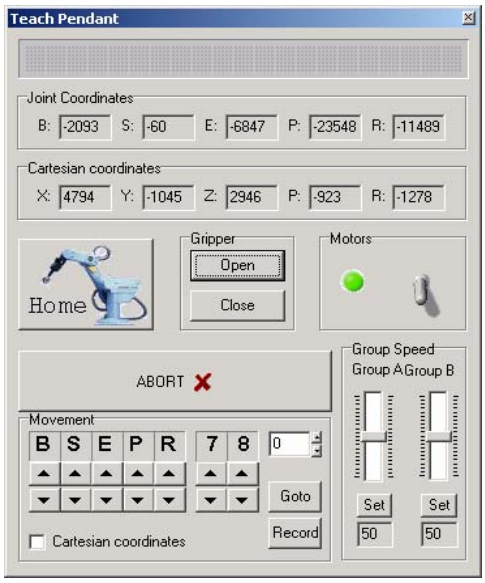


Figura 3.8 Janela do *teach pendant.*

Figura 3.9 Janela das entradas e saídas.

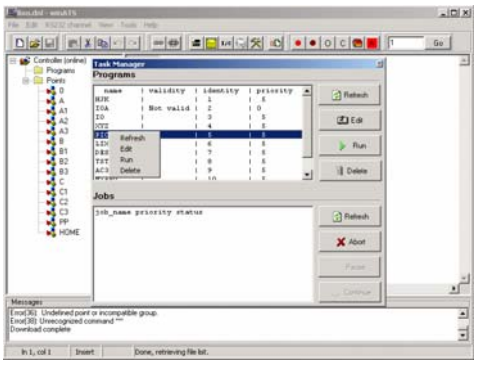


Figura 3.10 Janela do *Task Manager.*

As potencialidades de funcionamento em rede (protocolo TCP/IP) do "winRS232ROBOTcontrol" permitem a utilização remota das aplicações desenvolvidas para o robô *Scorbot ER VII*. Como mostra a figura 3.11, foi instalado um computador (PC servidor) junto ao controlador do robô *Scorbot ER VII* de forma a permitir o controlo e a monitorização remota do robô, através de computadores clientes ligados à rede. Como veremos no próximo capítulo, o controlo e a monitorização remota do robô *Scorbot ER VII* foram determinantes para a integração deste robô na CFF desenvolvida.

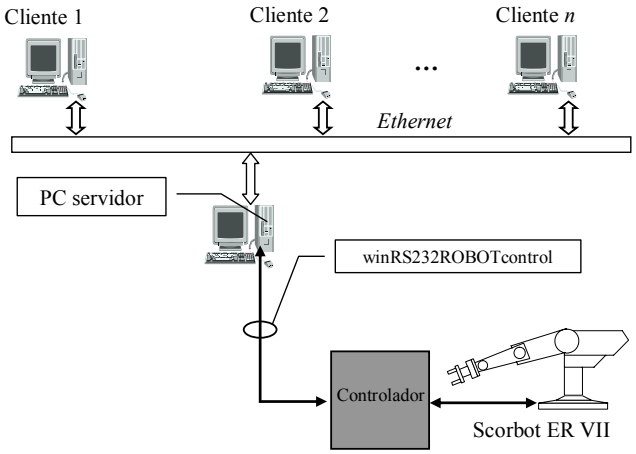


Figura 3.11 Controlo e monitorização remota do robô *Scorbot ER VII*.

3.4.4 Exemplo de aplicação do *winEthernetROBOTcontrol*

Com o objectivo de testarmos as potencialidades do "winEthernetROBOTcontrol", desenvolvemos uma aplicação industrial para uma empresa multinacional do ramo automóvel. Na aplicação industrial desenvolvida foi utilizado um robô industrial ABB IRB2400, uma prensa de corte, uma máquina de injecção, um transportador para recolha de *gitos*, uma máscara de sensores, um sistema de banho para arrefecimento das peças, e uma unidade de rejeição. O robô industrial ABB utilizado está equipado com um controlador *S4Cplus* e utiliza o Sistema Operativo (SO) *BaseWare OS 3.2* [ABB, 2003a], [ABB, 2003b] e [ABB, 2003c]. A finalidade da aplicação industrial desenvolvida é produzir uma grande variedade de peças para o fabrico de chaves de automóveis das mais variadas marcas.

A figura 3.12 apresenta o *layout* da aplicação industrial desenvolvida. Sucintamente, o ciclo de produção da aplicação desenvolvida apresenta a seguinte sequência:

1. O robô ABB IRB2400 aguarda autorização da máquina de injecção para retirar as peças do molde.
2. O robô posiciona as peças retiradas da máquina de injecção numa máscara de sensores, com o intuito do sistema verificar se as peças estão em conformidade com as condições exigidas. Se o conjunto de sensores detectar alguma anomalia nas peças, estas são rejeitadas e o sistema entra em alarme, não permitindo, por razões de segurança, a continuação da injecção de peças na máquina de injecção.
3. Se o conjunto de sensores não detectar anomalias nas peças, estas são mergulhadas num líquido de forma a provocar o arrefecimento das mesmas.
4. O robô aguarda autorização para colocar as peças na prensa de corte (SPT 8). Logo que as peças estejam devidamente posicionadas na prensa de corte, esta procede à separação das peças do *gito* e à respectiva limpeza das aparas resultantes do processo de injecção. O *gito* é reintroduzido no processo de fundição, através de um transportador utilizado para o efeito.

A substituição do molde utilizado na máquina de injecção (FRECH 200) permite o fabrico de peças para um grande número de marcas de automóveis. No entanto, sempre que se troca o molde na máquina de injecção é necessário proceder a ajustes e afinações na prensa de corte, uma vez que as peças variam de acordo com o modelo e a marca do automóvel. Como exemplo, a figura 3.13 (a) mostra a fotografia de duas peças, unidas por um *gito*, à saída da máquina de injecção. As figuras 3.13(b) e 3.13(c) mostra duas peças utilizadas em automóveis de marcas

diferentes. De referir que, as peças produzidas, na aplicação industrial por nós desenvolvida, são utilizadas nos sistemas de ignição dos automóveis. A figura 3.13(d) mostra o aspecto final desses sistemas de ignição.

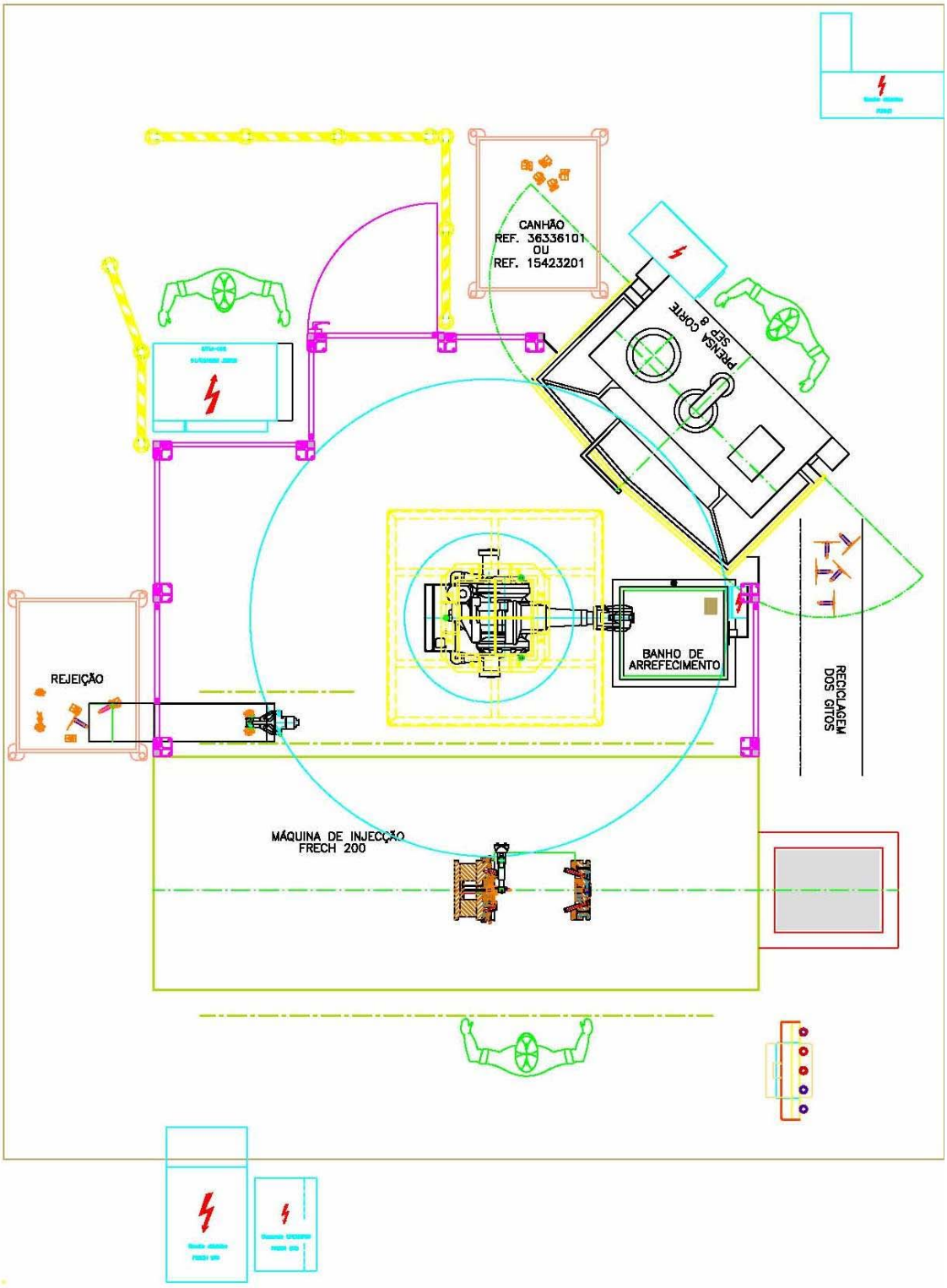
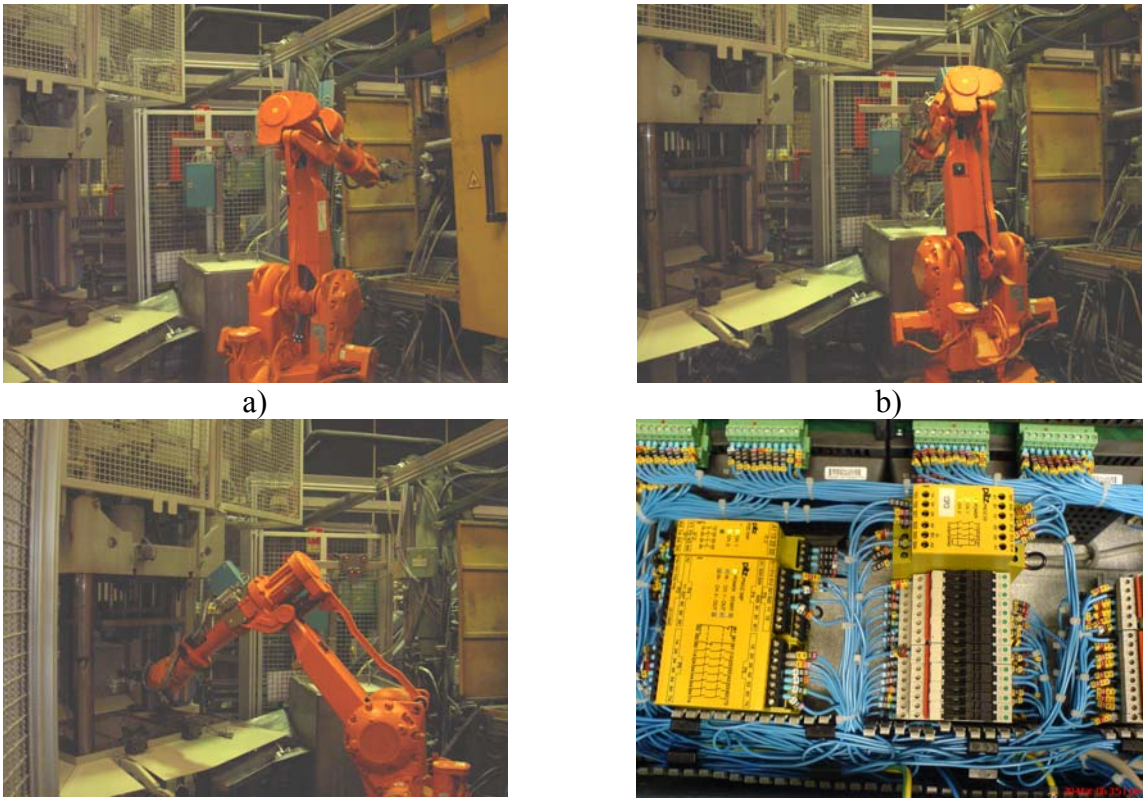


Figura 3.12 *Layout* da aplicação industrial desenvolvida.

a) b) c) d)

Figura 3.13 Fotografias com exemplos de peças produzidas na aplicação industrial desenvolvida.

As fotografias apresentadas na figura 3.14 mostram a aplicação industrial desenvolvida, onde é possível visualizar o robô utilizado ABB IRB2400. Na figura 3.14(a) é possível visualizar o robô a retirar as peças do molde da máquina de injecção, a figura 3.14(b) mostra o robô a efectuar o arrefecimento das peças e a figura 3.14(c) apresenta o robô a colocar as peças na prensa de corte. Inicialmente, a máquina de injecção e a prensa de corte não reuniam as condições necessárias para a respectiva integração e controlo na aplicação industrial que se pretendia desenvolver. Para tal, foi necessário efectuar alterações nas máquinas de forma a estas permitirem a integração e o controlo através de uma interface robótica. A fotografia apresentada na figura 3.14(d) mostra parte da interface robótica desenvolvida, e que se encontra no interior do controlador do robô ABB IRB2400, onde chegam e partem todos os sinais digitais necessários ao controlo e monitorização da aplicação industrial desenvolvida.



a) b)

c) d)

Figura 3.14 Fotografias do robô industrial ABB IRB2400.

As potencialidades de funcionamento em rede (protocolo TCP/IP) do "winEthernetROBOTcontrol" permitem o desenvolvimento de aplicações industriais de monitorização e controlo remoto de robôs industriais. Como mostra a figura 3.15, foi instalado um computador (PC servidor) junto ao controlador do robô ABB IRB2400 e dois computadores (cliente 1 e cliente 2) nos gabinetes de engenharia da empresa, com o objectivo de permitir o controlo e a monitorização remota da aplicação industrial desenvolvida. Através dos computadores instalados (PC servidor, cliente 1 e cliente 2) é possível efectuar operações, tais como:

1. Carregamento de programas no controlador do robô ABB IRB2400, a partir de qualquer um dos computadores.
2. Gestão de programas e ficheiros, incluindo ficheiros de registo da produção. Os ficheiros de registo de produção contêm informação variada, como por exemplo: tempos de ciclo, número de peças produzidas, número de peças rejeitadas, tempos de paragem, situações de alarmes ocorridas, entre outras.
3. Monitorização do estado de funcionamento dos equipamentos usados (máquina de injecção, prensa de corte e robô ABB IRB2400).
4. Recuperação de situações de paragens, colocando o robô ABB IRB2400 em posições de segurança. Apesar das recuperações de situações de paragem terem sempre em consideração a posição actual do estado do robô, estas devem ser feitas na presença do robô, ou seja, no PC servidor que se encontra junto ao robô.
5. Aproveitamento das potencialidades avançadas do controlador do robô ABB IRB2400, nomeadamente a capacidade de programação local através da linguagem RAPID (linguagem de programação dos robôs ABB).

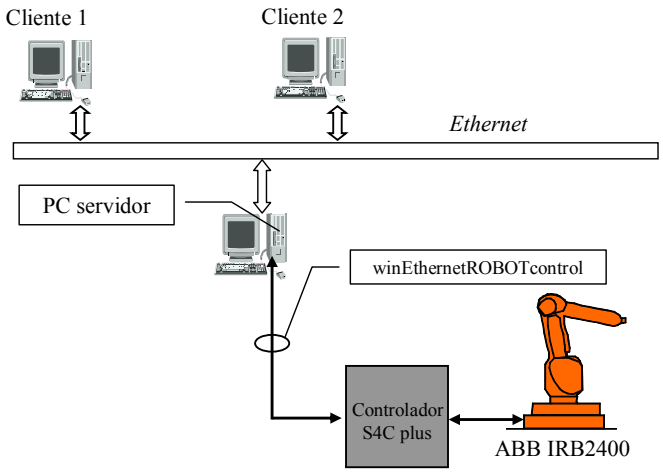


Figura 3.15 "winEthernetROBOTcontrol" na aplicação industrial desenvolvida.

A monitorização e o controlo da máquina de injecção e da prensa de corte são efectuados através da interface robótica desenvolvida. Assim, sempre que ocorre uma mudança de estado e/ou uma anomalia numa das máquinas, estas são imediatamente comunicadas ao controlador do robô ABB IRB2400 através da interface robótica. Desta forma, é possível monitorizar e controlar estas máquinas através do programa desenvolvido para o PC servidor, cliente 1 e cliente 2. A figura 3.16 apresenta o programa usado nos computadores (PC servidor, cliente 1 e cliente 2) da aplicação industrial desenvolvida, embora possa ser usado noutras aplicações industriais.

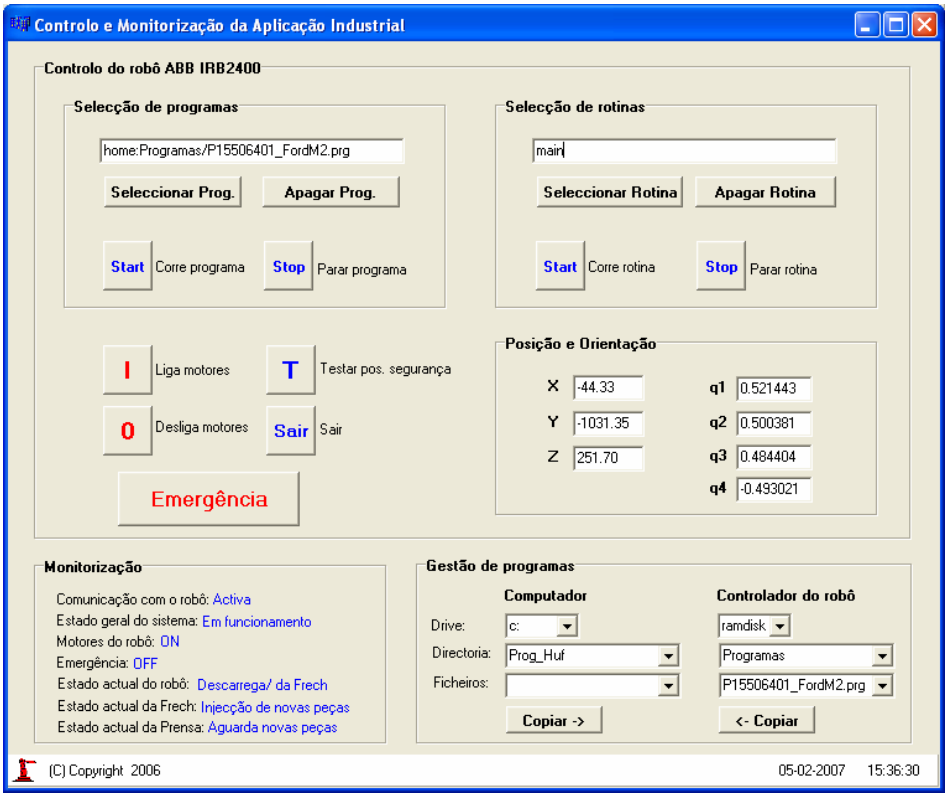


Figura 3.16 Programa usado na aplicação industrial desenvolvida.

O exemplo da aplicação industrial apresentado demonstra claramente a aplicabilidade da arquitectura da ferramenta de software "winEthernetROBOTcontrol" no desenvolvimento de aplicações de controlo e monitorização de robôs industriais, podendo estes estar integrados com os mais variados equipamentos industriais.

3.5 Software para máquinas CNC industriais

A maioria das máquinas CNC actualmente utilizadas na indústria permite a sua ligação a redes *Ethernet*. Porém, as actuais máquinas CNC ainda apresentam as seguintes limitações [Ferrolho *et al.*, 2005] e [Ferrolho e Crisóstomo, 2007a]:

1. As máquinas CNC industriais possuem controladores numéricos (NC - *Numerical Control*) fechados, estando estas máquinas vocacionadas para trabalharem individualmente, e não integradas com outros equipamentos (*e.g.*, robôs manipuladores, outras máquinas CNC, entre outros).
2. Actualmente os controladores das máquinas CNC industriais não permitem o seu controlo remoto a partir de um computador externo.
3. As máquinas CNC industriais não possuem ambientes de programação suficientemente poderosos para o desenvolvimento de tarefas que necessitem de técnicas de controlo complexas (*e.g.*, integração e controlo de máquinas CNC em CFF).
4. Máquinas CNC de diferentes fabricantes têm linguagens e ambientes de programação diferentes, tornando a integração e o controlo das mesmas difícil.

Com a finalidade de reduzir estas limitações nas máquinas CNC industriais, foram desenvolvidas ferramentas de software que permitem o controlo das máquinas CNC através de controlo numérico directo (DNC – *Direct Numerical Control*). Para os tornos CNC foi desenvolvido o "winTURNcontrol" e para as fresadoras CNC foi desenvolvido o "winMILLcontrol". Com as ferramentas de software desenvolvidas, é possível reduzir muitas das actuais limitações existentes nas máquinas CNC [Ferrolho *et al.*, 2005] e [Ferrolho e Crisóstomo, 2007a]:

1. É possível controlar e monitorar remotamente todas as funcionalidades das máquinas CNC (*e.g.*, abrir e fechar portas, abrir e fechar a prensa, abrir e fechar o torno, início de maquinagem, entre outras). Este controlo e monitorização remoto é muito importante para a integração das máquinas CNC nas modernas CFF.
2. É possível transferir e carregar programas de NC para as máquinas CNC, a partir de qualquer computador de uma rede *Ethernet*.
3. É possível desenvolver uma arquitectura distribuída de software, baseada em computadores pessoais, usando redes *Ethernet*.
4. É possível desenvolver aplicações que permitam a exploração remota das máquinas CNC usando sempre a mesma linguagem de programação (*e.g.*, C++).
5. É possível utilizar computadores pessoais como meio de programação, tirando partido de um grande número de ferramentas de programação e análise disponíveis nos mesmos.
6. É possível desenvolver GUI para máquinas CNC, de forma simples e rápida.

Actualmente, as máquinas CNC podem vir equipadas de fábrica com interfaces robóticas para permitir o seu controlo através de entradas e saídas digitais. No entanto, em muitas das

aplicações industriais, a interface robótica não é suficiente por duas razões: primeiro, não permite o controlo de todas as funcionalidades das máquinas, como por exemplo, seleccionar e transferir programas de controlo numérico (código NC) para as máquinas; segundo, requer um elevado número de fios (um fio por cada função a controlar).

Alguns controladores de máquinas CNC (*e.g.*, Fanuc) só permitem ligações externas através da porta série RS232 [Ferrolho *et al.*, 2005]. Com o intuito de ultrapassar esta limitação, foi ligado um cabo *null modem* entre a COM1 e COM2 de cada computador que controla a respectiva máquina, como se mostra na figura 3.17. Os programas DNC desenvolvidos utilizam a COM1 para o envio de ordens (*e.g.*, abrir a porta, fechar a prensa, iniciar a maquinagem, entre outras) para as máquinas, e os controladores das máquinas CNC recebem essas mesmas ordens através da COM2. Para isso, é necessário configurar o software das máquinas CNC de forma a estas poderem receber as ordens através da COM2. Com o intuito de podermos controlar remotamente as máquinas, foram desenvolvidos servidores *Ethernet* para serem utilizados nos computadores que controlam as máquinas CNC, como mostra a figura 3.17. Os servidores *Ethernet* recebem a informação através da rede *Ethernet* e de seguida enviam essa mesma informação através da porta série, para os respectivos controladores das máquinas. Por exemplo, se o cliente 1, na figura 3.17, precisar de enviar uma ordem (*e.g.*, fechar porta) para a máquina A, o servidor *Ethernet* dessa máquina irá receber essa ordem através da rede *Ethernet*. De seguida, o servidor envia a ordem para a COM1 e o controlador da máquina CNC receberá a ordem através da COM2 (cabo *null modem*), e por conseguinte a máquina A fecha a porta.

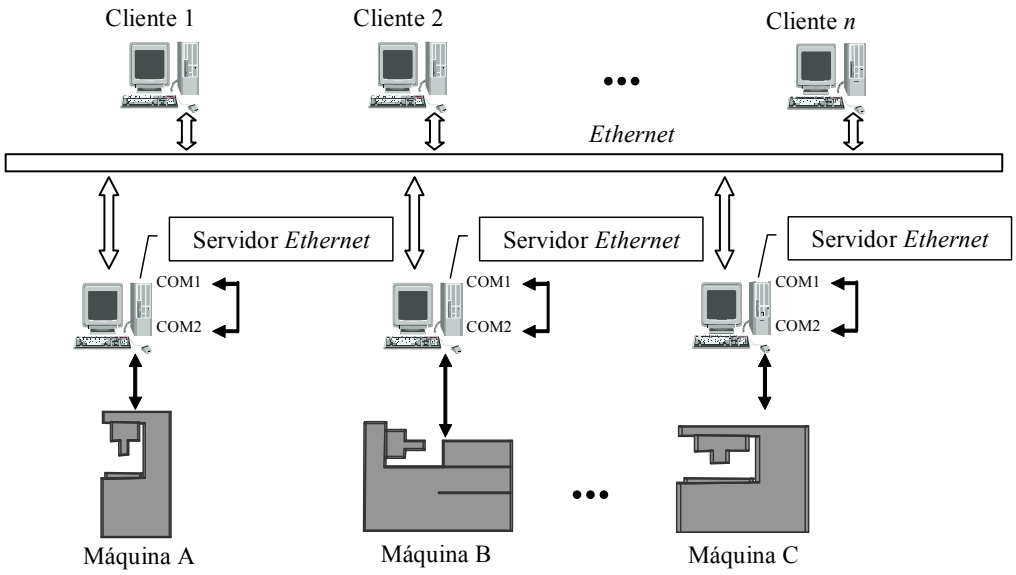


Figura 3.17 Controlo de máquinas CNC através de uma rede *Ethernet*.

As interfaces DNC desenvolvidas permitem que um computador controle e monitorize uma ou mais máquinas CNC. A interface DNC estabelece uma ligação entre o computador cliente (*e.g.*, cliente1) e o computador da máquina CNC. Depois de activado o modo DNC, os computadores clientes podem controlar e monitorizar as máquinas CNC através da rede *Ethernet*, como mostra a figura 3.17. A interface DNC desenvolvida estabelece uma ligação TCP/IP entre o computador cliente e o computador da máquina CNC. Com a interface DNC desenvolvida é possível remotamente: transferir ficheiros com código NC (programas de maquinagem) para as máquinas, trabalhar com blocos diferentes dos ficheiros NC, visualizar alarmes e informação relativa ao funcionamento da máquina, e ainda, controlar todos os dispositivos das máquinas CNC (*e.g.*, portas, posições das ferramentas, motores, sistema de refrigeração, entre outros) [Ferrolho *et al.*, 2005] e [Ferrolho e Crisóstomo, 2007a].

3.5.1 Descrição da interface DNC desenvolvida

A interface DNC desenvolvida estabelece uma ligação TCP/IP entre o computador cliente (PC cliente, computador da produção, computador CFF, etc.) e o computador que controla a máquina CNC (PC servidor), como mostra a figura 3.17. Após estabelecida a ligação DNC, o computador cliente pode controlar e monitorar a máquina CNC, através do PC servidor. Todos os dados relativos aos programas NC, ferramentas a utilizar na maquinagem, zeros das peças que vão ser maquinadas, entre outros, podem ser transferidos do computador cliente para a máquina CNC, ou seja, o computador cliente pode assegurar todo o controlo da produção. Desta forma, com a interface DNC desenvolvida é possível remotamente:

- Transferir ficheiros com código NC para as máquinas CNC;
- Operar ficheiros NC de diferentes peças;
- Visualizar alarmes e informações sobre o estado das máquinas CNC;
- Controlar todos os dispositivos das máquinas CNC, como por exemplo: portas, sistema de refrigeração, prensa e bucha de aperto, entre outros.

Como já foi referido anteriormente, a comunicação entre o computador cliente e o computador que controla a máquina CNC é efectuada através de uma ligação TCP/IP. A tabela 3.4 apresenta o formato do pacote de comunicação DNC utilizado. O campo *CheckSum* é fundamental na detecção de erros durante a transmissão de um pacote de comunicação. Se este campo não for preenchido correctamente, a máquina não reconhece a validade do pacote de comunicação e ignora-o.

Tabela 3.4 Formato do pacote de comunicação DNC

Campo	Bytes	Comentário
CheckSum	1, binário	O *checksum* é formado pela adição do número de bytes do cabeçalho (excluindo o campo de *checksum*) com o número de bytes de dados.
Grupo de comandos	1, ASCII	Código correspondente ao grupo do comando a enviar.
Identificação do comando	1, ASCII	Código correspondente ao comando a enviar.
Número do pacote	1, binário	O número do pacote serve para identificar a ordem dos pacotes em comandos constituídos por vários pacotes (*e.g.*, transmissão de programas). O 1.º pacote tem o número 1. O último pacote tem sempre o número 69 ('E' em ASCII). Deste modo um comando só pode ter no máximo 69 pacotes.
Número da mensagem	2, binário	Os pacotes têm sempre números consecutivos para permitir a identificação de pacotes de dados em caso de erro.
Tamanho dos dados	2, binário	Indica o número de bytes que vão ser enviados no campo de dados.

Em caso de erros durante a transmissão, ou se forem enviados comandos impossíveis de serem executados pela máquina CNC, serão enviadas mensagens de erro de comunicação ao PC cliente, como mostra a tabela 3.5. Quando ocorre um destes erros, o pacote enviado para a máquina CNC é rejeitado.

Tabela 3.5 Erros de comunicação

Dados (ASCII)	Significado
1	Erro geral de comunicação
2	Comando desconhecido
3	Erro no *CheckSum*
4	Comando impossível de executar
5	Pacote incompleto

O erro "comando impossível de executar" ocorre quando o comando enviado para a máquina CNC não puder ser executado durante o actual estado da mesma. Normalmente, isto acontece quando um novo comando é enviado para a máquina antes do último comando ter sido confirmado pela máquina. Este tipo de erro também ocorre quando é enviado um comando para a máquina sem esta ter o modo DNC activo, ou quando o pacote exceder 256 *bytes*. O erro "pacote incompleto" ocorre quando a máquina não recebe o pacote completo e os dados em falta não chegam dentro de um determinado período (*timeout*) [Ferrolho *et al.*, 2005] e [Ferrolho e Crisóstomo, 2007a].

A tabela 3.6 apresenta alguns dos comandos desenvolvidos para a interface DNC. Na referida tabela, o símbolo → significa o envio de um pacote do PC cliente para o PC servidor (PC da máquina) e o símbolo ← significa o envio de um pacote do PC servidor para um PC cliente (*feedback* da máquina). Para que uma máquina CNC execute um comando remotamente é necessário que um PC cliente envie uma mensagem (conjunto de caracteres – pacote) para o PC

servidor da máquina. A mensagem é formada pelo comando e pelos parâmetros: COMANDO|PARAMETROS. O PC servidor responde com duas mensagens: a primeira, indica ao PC cliente que o comando foi recebido e irá ser executado; a segunda, indica ao PC cliente se o comando foi bem executado ou se ocorreu algum erro.

Tabela 3.6 Alguns comandos desenvolvidos para a interface DNC

Direcção	Mensagem	Significado
→	START_DNC|_	Solicita a operação em modo DNC
←	START_DNC|WAIT	Em execução
←	START_DNC|ACK	Modo DNC activado
→	TERMINATE_DNC|_	Termina a operação em modo DNC
←	TERMINATE_DNC|WAIT	Em execução
←	TERMINATE_DNC|ACK	Comando executado
→	DOORS|OPEN	Abre a porta
←	DOORS|WAIT	Em execução
←	DOORS|OPEND	Comando executado
→	DOORS|CLOSE	Fecha a porta
←	DOORS|WAIT	Em execução
←	DOORS|CLOSED	Comando executado
→	CLAMPING_DEVICE|OPEN	Abre o dispositivo de aperto
←	CLAMPING_DEVICE|WAIT	Em execução
←	CLAMPING_DEVICE|OPEND	Comando executado
→	CLAMPING_DEVICE|CLOSE	Fecha o dispositivo de aperto
←	CLAMPING_DEVICE|WAIT	Em execução
←	CLAMPING_DEVICE|CLOSED	Comando executado
→	BLOW_OUT|OFF	Desliga o "sopro" de ar comprimido
←	BLOW_OUT|WAIT	Em execução
←	BLOW_OUT|OFFD	Comando executado
→	BLOW_OUT|ON	Liga o "sopro" de ar comprimido
←	BLOW_OUT|WAIT	Em execução
←	BLOW_OUT|OND	Comando executado
→	SLEEVE|BACKWARD	Movimenta o "*Sleeve*" para trás
←	SLEEVE|WAIT	Em execução
←	SLEEVE|BACKWARDD	Comando executado
→	SLEEVE|FORWARD	Movimenta o "*Sleeve*" para a frente
←	SLEEVE|WAIT	Em execução
←	SLEEVE| FORWARDD	Comando executado
→	SWIVEL|_	Roda o porta-ferramentas
←	SWIVEL|WAIT	Em execução
←	SWIVEL|ACK	Comando executado
→	REFERENCING|_	Encontra o ponto de referência dos eixos
←	REFERENCING|WAIT	Em execução
←	REFERENCING|ACK	Comando executado
→	PROGRAM_SELECTION|number	Abre um programa
←	PROGRAM_SELECTION|WAIT	Em execução
←	PROGRAM_SELECTION|ACK	Comando executado
→	START_RELEASE|_	Executa o programa NC
←	START_RELEASE|WAIT	Em execução
←	START_RELEASE|ACK	Comando executado

Sucintamente, a interface DNC desenvolvida permite o controlo e monitorização remota de máquinas CNC. Actualmente, este tipo de interface é extremamente importante para a integração e controlo de máquinas CNC em sistemas de fabrico, como por exemplo, SFF e CFF.

3.5.2 Exemplo de aplicação da interface DNC

A interface DNC desenvolvida foi aplicada a duas máquinas CNC que temos no nosso laboratório, uma fresadora EMCO Mill 155 e um torno EMCO Turn 55. Ambas as máquinas possuem um controlador *GE Fanuc 21* e uma interface robótica. Para muitas das aplicações, e como já foi referido anteriormente, esta interface robótica é por vezes insuficiente.

Foram desenvolvidos dois programas de software em *C++ Builder* que permitem, através da interface DNC desenvolvida, o controlo das duas máquinas CNC que temos no laboratório. Estes programas permitem o controlo remoto de todas as funcionalidades das máquinas CNC. Para o torno foi desenvolvido o programa "winTURNcontrol" e para a fresadora o "winMILLcontrol". A figura 3.18 apresenta os programas desenvolvidos para as duas máquinas. Estes programas permitem dois modos de operação:

- Manual – neste modo é possível o controlo manual das máquinas, através da interacção com os botões disponibilizados no programa.
- Remota – os programas desenvolvidos possuem um servidor de forma a permitir o controlo remoto do torno e da fresadora.

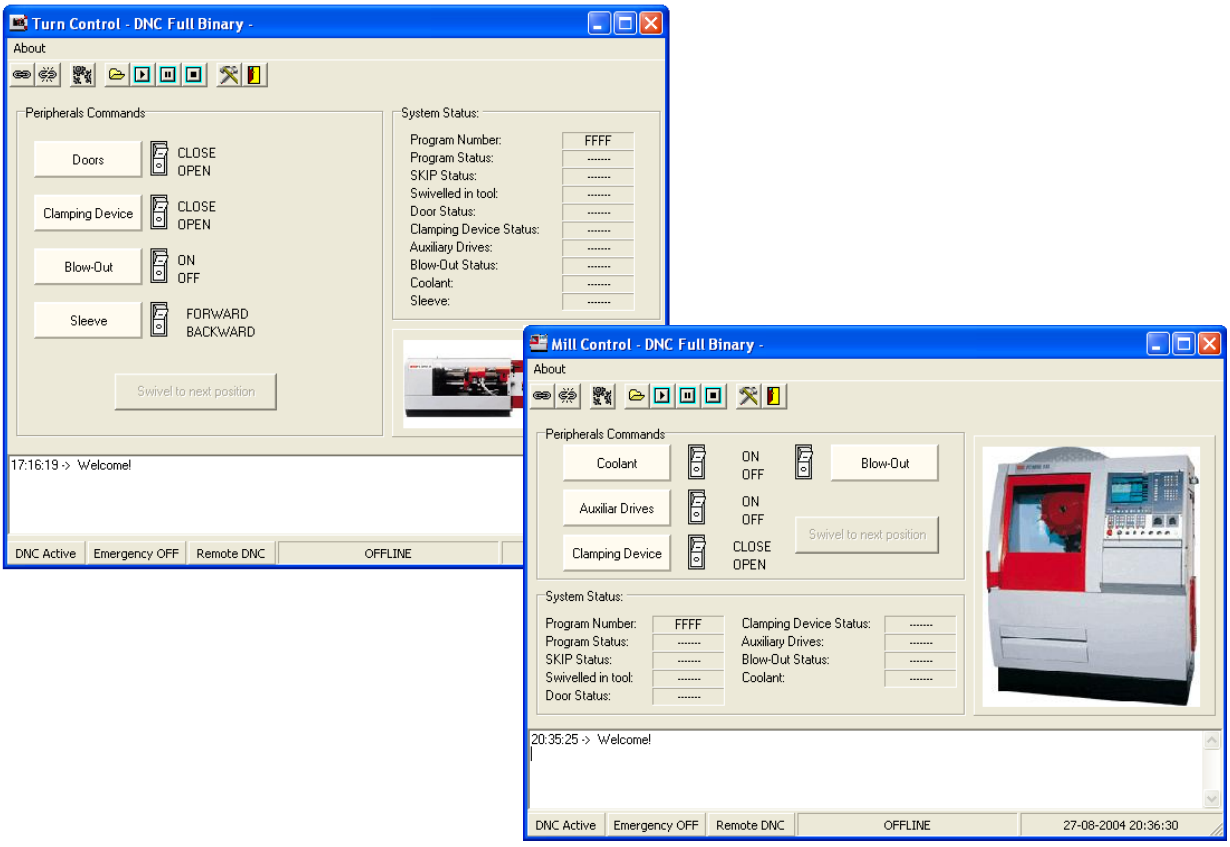


Figura 3.18 Programas desenvolvidos para o torno e para a fresadora.

Como mostra a figura 3.19, os programas desenvolvidos permitem a um computador (PC cliente) controlar e monitorar as duas máquinas CNC que temos no nosso laboratório. Para isso, o PC cliente deve estabelecer as respectivas ligações com os PC servidores de cada máquina. Depois de activado o modo DNC, o servidor de cada máquina recebe os pedidos pela rede *Ethernet* e, de seguida, envia-os para a máquina através da porta série (cabo *null modem*) [Ferrolho *et al.*, 2005] e [Ferrolho e Crisóstomo, 2007a].

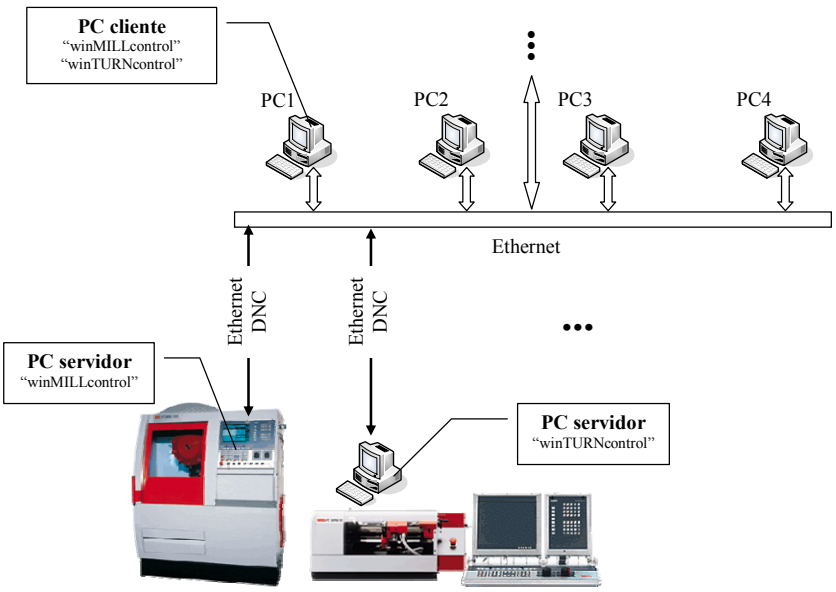


Figura 3.19 Controlo e monitorização da fresadora e do torno.

3.6 Software e Hardware USB

Desenvolvemos uma ferramenta de software e hardware USB a que chamamos "USB software" e "USB kit", respectivamente. A finalidade destas ferramentas é permitir a integração e o controlo de transportadores, motores, sensores, actuadores, entre outros, em SFF e CFF. O hardware desenvolvido "USB kit" possui várias entradas e saídas onde é possível ligar vários sensores e actuadores de diferentes fabricantes. Como mostra a figura 3.20, a comunicação entre o PC1 e o hardware "USB kit" é feita através duma porta USB. Por sua vez, o PC1 tem a possibilidade de comunicar com os outros computadores através da rede *Ethernet* [Ferrolho e Crisóstomo, 2007a].

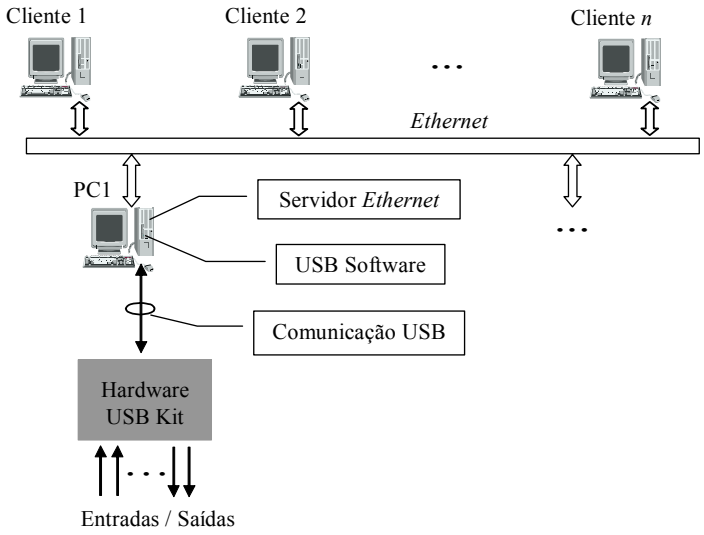


Figura 3.20 Software USB e kit USB.

3.6.1 Exemplo de aplicação do software e hardware USB

Como exemplo de aplicação do software e hardware USB referimos o sector de transporte, desenvolvido e implementado na CFF descrita no próximo capítulo. Como veremos nesse capítulo, o controlo e a monitorização do transportador utilizado no sector de transporte da CFF só foi possível devido às potencialidades do software e hardware USB desenvolvidos (protocolo TCP/IP).

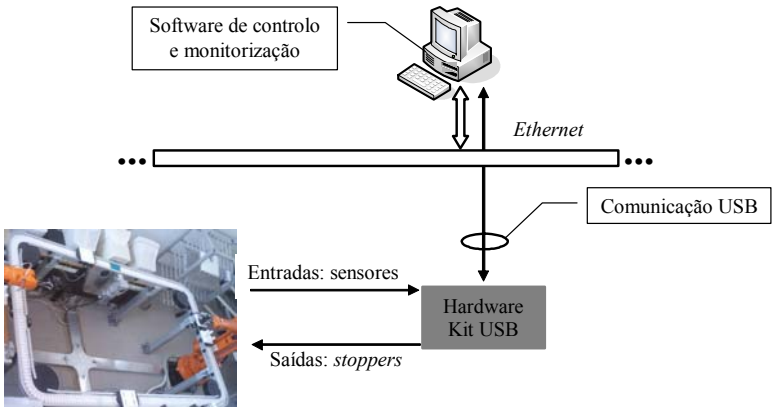


Figura 3.21 Controlo e monitorização de um transportador.

3.7 Resumo do Capítulo

Neste capítulo apresentaram-se e discutiram-se, longamente, muitos dos problemas associados ao controlo de robôs manipuladores industriais e máquinas CNC industriais. Foram desenvolvidas várias ferramentas de software para robôs manipuladores industriais e máquinas CNC com o objectivo de permitir a integração, coordenação e controlo destes equipamentos em SFF e CFF.

Actualmente, as máquinas CNC e os robôs manipuladores continuam a ser máquinas difíceis de programar, tendo em conta que é necessário dominar vários ambientes de programação. Cada fabricante possui um ambiente de programação próprio (geralmente local). Normalmente, cada fabricante desenvolve um controlador para a sua estrutura mecânica e uma linguagem de programação com o respectivo interpretador para serem usados com esse controlador. As linguagens não são normalmente compatíveis, apesar de estruturalmente serem muito próximas, e exigem algum treino e experiência para serem usadas eficientemente. Esta prática generalizada explica a enorme confusão de linguagens e ambientes de programação actualmente existentes no mercado.

Ao longo deste capítulo, foram abordados aspectos relacionados com a utilização de robôs industriais e máquinas CNC, nomeadamente software de monitorização, integração e controlo. Foram, também focadas, neste capítulo, questões de acesso e comando remoto desses equipamentos. A filosofia adoptada pode ser usada noutros equipamentos de uma CFF, com o objectivo de desenvolver uma arquitectura que permita uma programação integrada de toda a célula. As potencialidades de funcionamento em rede (protocolo TCP/IP) das ferramentas de software desenvolvidas permitem a integração e o controlo de robôs industriais, de máquinas CNC, de transportadores, entre outros, em SFF e CFF.

No âmbito deste capítulo, foram desenvolvidas e apresentadas as ferramentas de software: "winRS232ROBOTcontrol", "winEthernetROBOTcontrol", "winMILLcontrol", "winTURN--control" e USB software. Estas ferramentas de software permitem a integração, o controlo e a monitorização de robôs e máquinas CNC industriais, em sistemas industriais de produção. As ferramentas de software desenvolvidas e apresentadas neste capítulo foram utilizadas com sucesso nos robôs e máquinas CNC que temos no laboratório (robôs: ABB IRB140, ABB IRB1400 e Scorbot ER VII; máquinas CNC: EMCO MILL 155 e EMCO TURN 55).

A arquitectura de software desenvolvida pode ser extensível a outros robôs e máquinas CNC de outros fabricantes. Porém, como é facilmente compreensível, será necessário realizar algumas alterações (*upgrades*) na arquitectura de software proposta. A ideia básica é definir uma biblioteca de funções para cada um dos equipamentos que permita o controlo e a monitorização remota dos mesmos.

Desenvolvemos uma CFF com a finalidade de testar as potencialidades das ferramentas de software desenvolvidas em aplicações industriais. As ferramentas de software desenvolvidas no âmbito deste capítulo foram determinantes para a integração, coordenação e controlo dos vários equipamentos utilizados na CFF. Esta CFF está descrita no capítulo 4.

Capítulo 4

Célula Flexível de Fabrico: desenvolvimento, coordenação, integração e controlo

"I hear and I forget. I see and I remember. I do and I understand."
Confucius

4.1 Introdução

Há décadas, a automatização era caracterizada por pequenas ilhas com operações automatizadas, onde o factor humano era fundamental como elemento integrador e sincronizador de todas as operações. Este tipo de automatização caracterizava-se, entre outros factores, por um elevado número de operários, uma grande existência de *stocks* e *layouts* não optimizados. Caminhou-se depois para soluções de automatização centralizada. Nestas, toda a informação é centralizada num único local, onde são tomadas todas as decisões e de onde partem todas as ordens. Consequentemente, os *layouts* foram melhorados, o número de operários foi bastante reduzido, continuando, no entanto, a existir um nível considerável de *stocks*. Após a década de 60, com o desenvolvimento e a utilização crescente de unidades de processamento de informação, as funções de condução dos processos foram sendo cada vez mais distribuídas pelo terreno e junto dos locais onde são necessárias, surgindo assim o que é actualmente designado por arquitecturas distribuídas ou por sistemas de controlo hierárquico distribuído. Este nível de automatização caracteriza-se por uma gestão global e integrada da informação; pela redução de *stocks* a níveis mínimos; pela inserção de máquinas de controlo numérico e de robôs manipuladores; pelo manuseamento automático de materiais; pela redução do número de operários (sendo em alguns sectores praticamente nulo na área directamente relacionada com a produção); pela utilização dos modernos conceitos de JIT (*Just-in-Time*) e TQM (*Total Quality Management*), e ainda, por uma utilização muito mais intensiva dos equipamentos. A integrar todos estes novos conceitos das tecnologias da produção, existe hoje a filosofia CIM (*Computer Integrated Manufacturing*). O conceito de CIM refere-se à completa automatização da fábrica, na qual todos os processos e actividades são controlados por computador, existindo em permanente circulação uma grande quantidade de informação proveniente de todos os sectores.

Um dos mais recentes desenvolvimentos na área da automação industrial é o conceito de Célula Flexível de Fabrico (CFF). As CFF são sistemas computorizados constituídos por vários tipos de equipamentos, usualmente ligados através de uma rede local (LAN) [Kusiak, 1986], [Waldner, 1992], [Cho *et al.*, 1995] e [Chang, 1991]. Actualmente, desenvolver uma CFF não é uma tarefa fácil, pois, normalmente, uma CFF usa equipamentos (*e.g.*, robôs, máquinas CNC, entre outros) de diferentes fabricantes, tendo cada um deles o seu ambiente e linguagem de programação. Assim, a integração e coordenação de robôs e máquinas CNC de diferentes fabricantes em CFF é uma tarefa difícil.

No âmbito deste capítulo, foi desenvolvida uma CFF com características industriais, com o objectivo de se estudarem estes sistemas de produção e de se testarem as ferramentas de software desenvolvidas, descritas no capítulo 3. A CFF desenvolvida tem uma estrutura de controlo hierárquica baseada em cinco níveis: engenharia, planeamento, sequenciamento, controlo e aquisição de dados. Cada nível comunica com aquele que lhe está imediatamente acima ou abaixo, não sendo permitida a comunicação na horizontal. A interligação de todos os equipamentos na CFF (quer de fabrico quer de processamento de informação) é feita através de uma rede de computadores *Ethernet*. A coordenação, integração e controlo da CFF só foram possíveis graças às ferramentas de software desenvolvidas e apresentadas no capítulo 3.

Este capítulo encontra-se organizado da seguinte forma: a secção 4.2 apresenta o *layout* e a estrutura hierárquica da CFF desenvolvida em laboratório; na secção 4.3 é feita uma apresentação detalhada dos quatro sectores (fabrico, montagem de peças, transporte e armazenamento) desenvolvidos para a CFF, bem como as ferramentas de software e protocolos de comunicação desenvolvidos para cada sector; a secção 4.4 apresenta o software desenvolvido para o computador central (gestor da CFF), assim como, as funções dos três primeiros níveis da estrutura hierárquica: engenharia, planeamento e sequenciamento. Por conseguinte, esta secção aborda o processo CAD/CAM/CNC e os problemas de sequenciamento inerentes à CFF; a secção 4.5 mostra como a CFF funciona num ambiente real de produção e, por último, na secção 4.6 encontra-se um resumo do trabalho desenvolvido neste capítulo.

4.2 Apresentação da Célula Flexível de Fabrico

Foi desenvolvida uma CFF com o intuito de se testarem as potencialidades das ferramentas de software desenvolvidas, apresentadas no capítulo 3, em aplicações industriais. A CFF é

constituída por quatro sectores, sendo o controlo e a monitorização dos equipamentos, de cada sector, realizado por quatro computadores: PC1 – sector da produção, PC2 – sector de montagem de peças, PC3 – sector de transporte, e PC4 – sector de armazenamento. A coordenação, sincronização e integração dos quatro sectores é feita pelo computador central – gestor da CFF. Assim, os sectores da CFF, que são controlados por computadores e software distintos, são [Ferrolho e Crisóstomo, 2007a]:

- O sector de fabrico, constituído por duas máquinas CNC (fresadora e torno), um robô ABB IRB140 e um *buffer*;
- O sector de montagem de peças, constituído por um robô *Scorbot ER VII*, um *conveyor* e uma mesa de montagem;
- O sector de transporte, constituído por um grande transportador rectangular;
- O sector de armazenamento, constituído por cinco armazéns e por um robô ABB IRB1400.

A figura 4.1 apresenta o *layout* da CFF desenvolvida, onde é possível visualizar os quatro sectores [Ferrolho e Crisóstomo, 2006d] e [Ferrolho e Crisóstomo, 2007a].

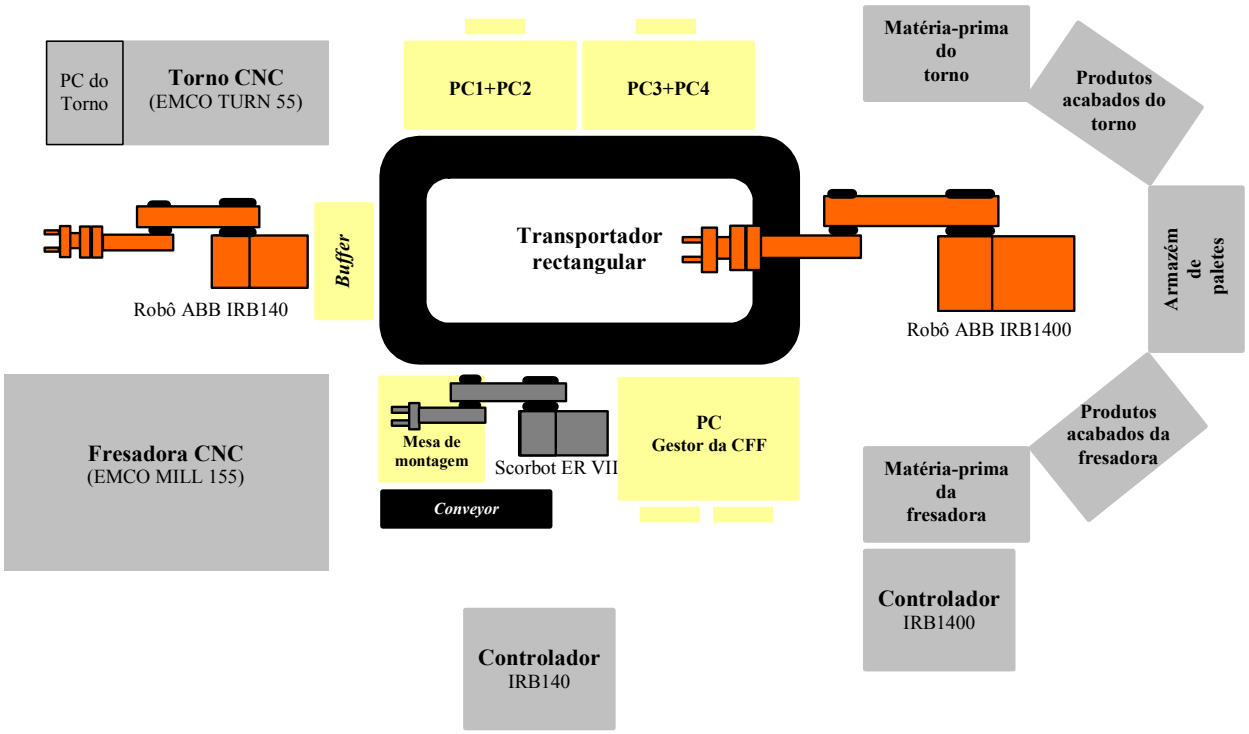
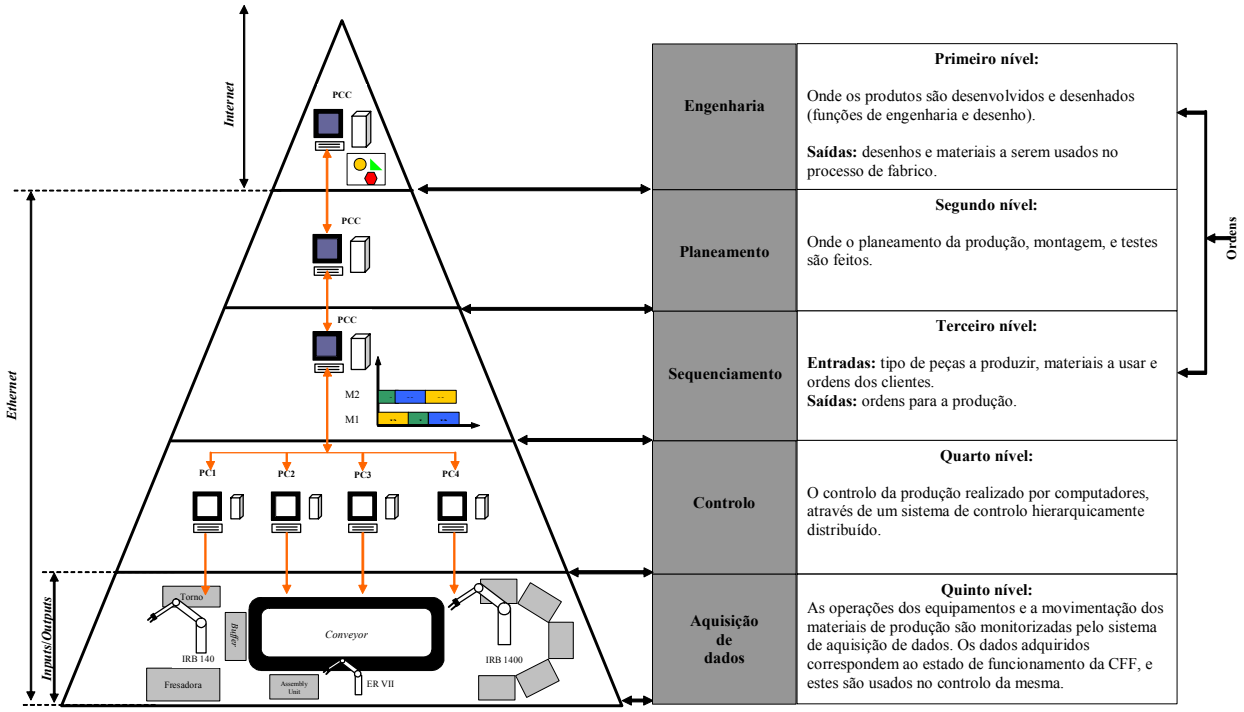


Figura 4.1 *Layout* da Célula Flexível de Fabrico.

Sempre que um computador necessite de comunicar com outro computador, é enviada uma mensagem através da rede *Ethernet*. As mensagens são compostas por um comando e por um

conjunto de parâmetros: COMANDO|PARAMETROS. O computador que recebe a mensagem (servidor) envia ao computador que a solicita (cliente) duas mensagens: a primeira, indica ao computador cliente que o comando foi recebido; a segunda, informa o computador cliente se o comando foi executado com sucesso ou se ocorreu algum erro.

A figura 4.2 apresenta a estrutura hierárquica implementada na CFF. Esta estrutura hierárquica é baseada no conceito de sistemas de controlo hierárquico distribuído. As acções de controlo são efectuadas pela junção de máquinas com poder de decisão, que estão geograficamente distribuídas, sendo perfeitamente autónomas e auto-contidas. Através da junção de esforços, estas máquinas trabalham para a implementação da tarefa de controlo global. O conjunto total das funções a serem implementadas pelo sistema de controlo hierárquico distribuído tem diferentes exigências ao nível da rapidez de actuação e da importância estratégica dessa mesma actuação. Assim, as acções a implementar surgem agrupadas em vários níveis hierárquicos, havendo características funcionais e temporais bem específicas para cada nível [Ferrolho e Crisóstomo, 2006b], [Ferrolho e Crisóstomo, 2006c] e [Ferrolho e Crisóstomo, 2007a].



PCC – Computador central (gestor da CFF): engenharia + planeamento + sequenciamento
PC1 – Sector de fabrico: Fresadora + Torno + Robô IRB140 + *buffer*
PC2 – Sector de montagem de peças: Robô *Scorbot ER VII* + *conveyor* + mesa de montagem
PC3 – Sector de transporte: transportador rectangular
PC4 – Sector de armazenamento: Robô IRB1400 + cinco armazéns

Figura 4.2 Estrutura hierárquica da CFF.

O computador central (gestor da CFF) é responsável pelos três primeiros níveis da estrutura hierárquica: engenharia, planeamento e sequenciamento. Este computador é o supervisor global da CFF, interactuando com os operadores humanos, com o exterior e com o nível imediatamente inferior da célula. Aqui, são tomadas todas as acções de controlo estratégico e é efectuada a supervisão global da CFF. As acções de controlo que decorrem no computador central são designadas por não críticas ou independentes do tempo, envolvendo constantes de tempo que podem ir até alguns minutos. Como exemplo, temos a acção relativa à informação dada ao operador sobre determinado tipo de ocorrência, ou a geração de relatórios de produção. As mensagens que flúem para o computador central são extensas (da ordem dos *Mbytes*) e possuem toda a informação do que se passa ao nível do processo de fabrico. Foi utilizada uma rede *Ethernet* para interligar o computador central com os restantes computadores utilizados na CFF. É também, a partir do computador central que se efectua a comunicação bidireccional com o exterior, utilizando para o efeito a *Internet*, pois esta é, actualmente, a rede mais usada na comunicação com os fornecedores e clientes.

Sucintamente, o primeiro nível (engenharia) contém as funções de engenharia e desenho, onde o produto é desenhado e desenvolvido, segundo as ordens dos clientes. Como saídas deste nível temos: os desenhos e as listas de materiais necessários ao desenvolvimento dos produtos. No segundo nível (planeamento) são feitos os planos de produção, planos de montagem e os testes. O terceiro nível (sequenciamento) tem como função gerar a sequência óptima de produção. O sequenciamento, consiste em determinar a sequência de passagem dos produtos pelos recursos, bem como definir a locação dos recursos, a partir da definição dos tempos de início e fim das operações, de modo a que, um ou mais critérios de desempenho sejam optimizados. Os planos de produção, juntamente com os desenhos dos produtos, a lista dos materiais necessários ao desenvolvimento dos mesmos, e as ordens dos clientes são as entradas do sequenciamento. As funções implementadas no terceiro nível são de tempo limitado, envolvendo constantes de tempo da ordem de alguns segundos. Estas, são funções não periódicas, e os dados que flúem de e para este nível são constituídos por mensagens com alguns *Kbytes* (*e.g.*, carregamento de programas de produção nas máquinas CNC).

O quarto nível (controlo) é constituído por um conjunto de computadores (PC1, PC2, PC3 e PC4) destinados a efectuarem o sincronismo entre as operações elementares do nível inferior (nível de aquisição de dados). Este nível efectua a supervisão, no que respeita à detecção de anomalias e à sua recuperação, enviando mensagens sobre o estado do funcionamento ao nível

imediatamente superior. As funções de controlo implementadas neste nível são designadas por funções de tempo crítico, não periódicas, envolvendo constantes de tempo da ordem de 20 ms a 100 ms. Os dados que flúem de e para este nível são constituídos por mensagens com o tamanho de alguns *bytes*. Os equipamentos utilizados neste nível estão ligados a uma rede *Ethernet*. A comunicação é efectuada sempre através de equipamentos de nível imediatamente superior ou inferior, e nunca, por equipamentos que se encontrem no mesmo nível hierárquico, ou seja, a comunicação é feita sempre na vertical.

O quinto nível (aquisição de dados) é o mais baixo da hierarquia, também designado por nível de instrumentação. Este nível integra todos os sensores e actuadores, encarregados de administrar o processo produtivo, e ainda, de tomar as medidas necessárias para uma correcta automação e supervisão. Aqui, são efectuadas as operações elementares de reacções aos estímulos do processo, também designadas por acções de controlo em tempo real. Neste nível, as mensagens são curtas (geralmente 1 *bit*) e em número muito elevado.

4.3 Sectores da CFF

A figura 4.3 apresenta com detalhe os quatro sectores desenvolvidos para a CFF. Ao nível destes sectores são utilizados três tipos de comunicação (ver figura 4.3): *Ethernet* no sector da produção e no sector de armazenamento, RS232 no sector de montagem de peças, e USB no sector de transporte. Os computadores PC1, PC2, PC3, PC4 e PC central (gestor da CFF) estão interligados através de uma rede *Ethernet*, como mostra a figura 4.3. Caso seja necessário, é possível desenvolver novas aplicações usando outros tipos de comunicação. A estrutura hierárquica utilizada na CFF permite acrescentar, de forma fácil e eficiente, novos sectores à mesma. Os novos sectores podem, também, usar equipamentos de diferentes fabricantes. Para isso, basta acrescentar um novo computador no nível quatro da estrutura hierarquia (controlo) e desenvolver ferramentas de software para controlo e monitorização dos equipamentos usados nesse novo sector. O desenvolvimento de software para o novo sector será feito com o recurso às bibliotecas de funções e às ferramentas de software desenvolvidas e apresentadas no capítulo 3. Para que o novo sector faça parte integrante da CFF, é necessário proceder à actualização do software no computador central (gestor da CFF) [Ferrolho e Crisóstomo, 2006b], [Ferrolho e Crisóstomo, 2007a] e [Ferrolho e Crisóstomo, 2007b].

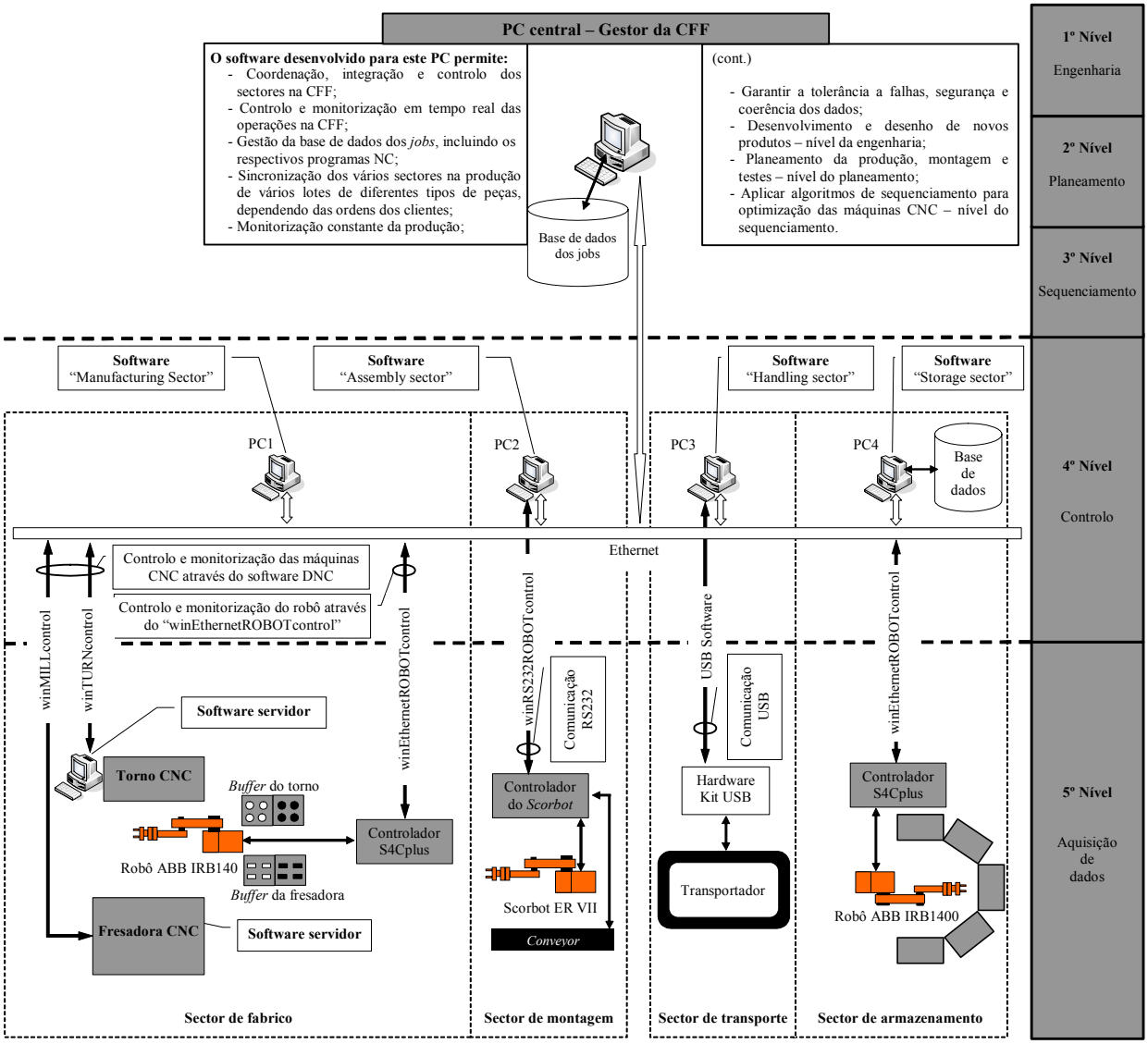


Figura 4.3 Sectores da CFF e estrutura hierárquica.

A seguir, apresenta-se uma descrição dos quatro sectores existentes na CFF. Estes sectores são controlados por quatro computadores (PC1, PC2, PC3 e PC4), estando estes posicionados no 4º nível (controlo) da estrutura hierárquica da CFF, como mostra a figura 4.3 [Ferrolho e Crisóstomo, 2007a].

4.3.1 Sector de fabrico

Como mostra a figura 4.4, o sector de fabrico é constituído por uma fresadora EMCO MILL 155 [Emco, 2001], um torno EMCO TURN 55 [Emco, 2002], um robô manipulador ABB IRB140 [ABB, 2003d] e um *buffer*. O *buffer* foi colocado no centro deste sector com o intuito de optimizar o processo de produção. Neste, estão disponíveis, em pequenas quantidades, matérias--primas para ambas as máquinas e são guardadas temporariamente as peças maquinadas nas máquinas.

O sector de fabrico é controlado e monitorizado através do software "Manufacturing Sector" instalado no PC1, como apresenta a figura 4.4. Este software foi desenvolvido em *C++ Builder* e apresenta as seguintes funções:

– Gestão e monitorização das matérias-primas e produtos acabados no *buffer*;
– Controlo e monitorização das máquinas CNC e do robô ABB IRB140;
– Coordenação, integração, sincronização e controlo dos equipamentos, e também de todas as tarefas desenvolvidas no sector de fabrico, de forma a produzir lotes variáveis de diferentes tipos de peças, consoante as ordens do computador central (gestor da CFF);
– Controlo e monitorização, em tempo real, do estado da produção;
– Garantir tolerância a falhas, segurança e coerência dos dados no sector de fabrico.

O controlo e a monitorização do robô ABB IRB140 são efectuados através das funções, procedimentos e eventos existentes em "winEthernetROBOTcontrol" (ver a tabela 3.2 do capítulo 3). O controlo e a monitorização da fresadora (EMCO MILL 155) e do torno (EMCO TURN 55) são efectuados pelas ferramentas de software DNC desenvolvidas e apresentadas no capítulo 3 – "winMILLcontrol" e "winTURNcontrol" [Ferrolho *et al.*, 2005] e [Ferrolho e Crisóstomo, 2007a]. Como mostra a figura 4.4, a interface robótica é usada em redundância com as ferramentas de software DNC.

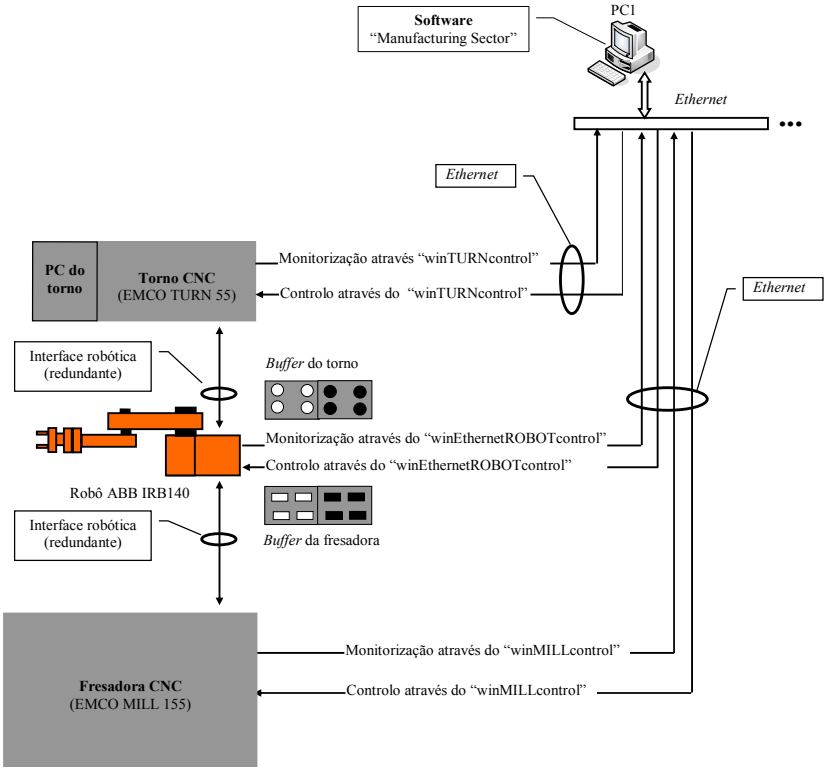


Figura 4.4 Controlo do sector de fabrico.

A tabela 4.1 apresenta as variáveis utilizadas no controlo do *buffer*. Estas variáveis são actualizadas automaticamente, sempre que é solicitado ao robô ABB IRB140 uma operação de carregamento ou descarregamento das máquinas CNC ou do *buffer*.

Tabela 4.1 Variáveis de controlo do *buffer*

Variável	Significado
BUFFER_RM_L	Matéria-prima do torno
BUFFER_RM_M	Matéria-prima da fresadora
BUFFER_FP_L	Produtos acabados do torno
BUFFER_FP_M	Produtos acabados da fresadora

A figura 4.5 mostra o software "Manufacturing Sector", desenvolvido para controlar e monitorizar o sector de fabrico. Este software está instalado no PC1 e tem incorporado um servidor de forma a permitir o controlo e a monitorização remota do sector de fabrico, através da rede *Ethernet*. A tabela 4.2 apresenta o protocolo de comunicação utilizado no sector de fabrico, onde o símbolo → significa o envio de uma mensagem do PC cliente (PC1) para o PC servidor (PC do robô ABB IRB140 / PC da fresadora / PC do torno), e o símbolo ← significa o envio de uma mensagem do PC servidor para um PC cliente (*feedback* do robô / fresadora / torno).

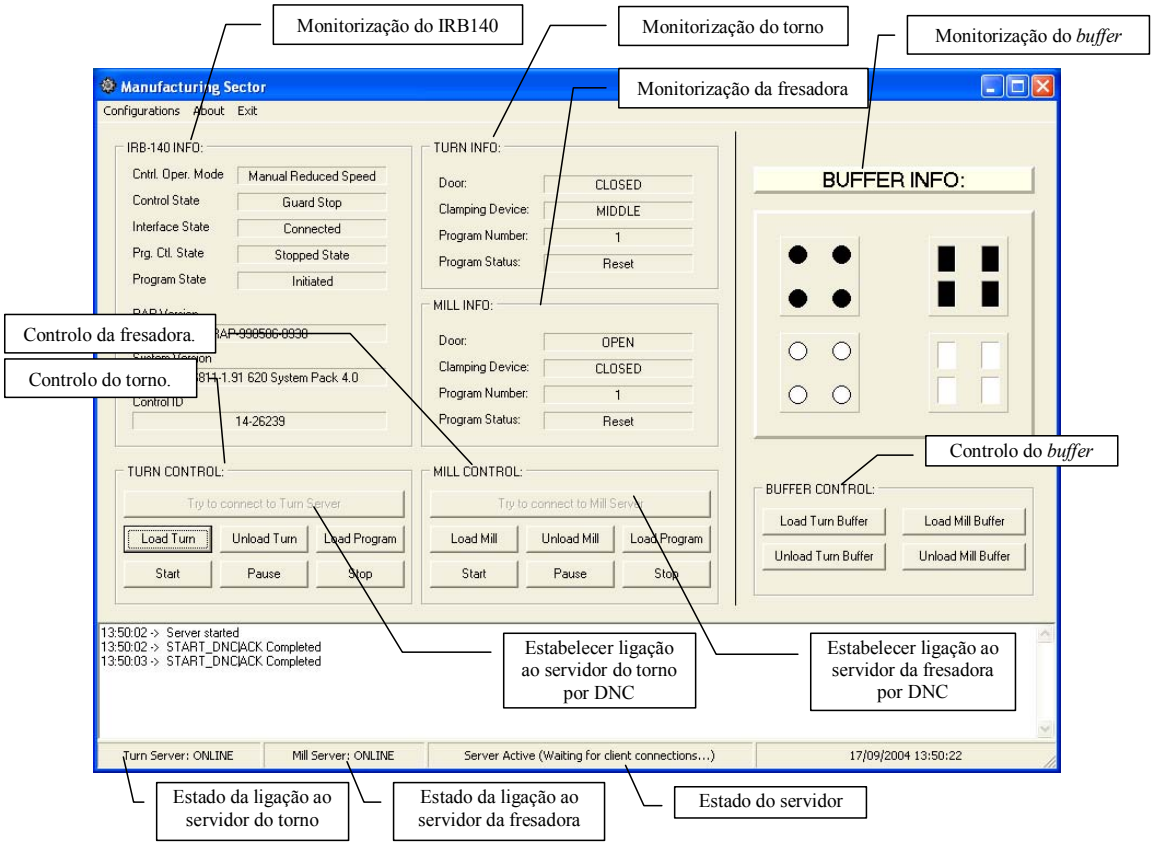


Figura 4.5 Software "Manufacturing Sector" desenvolvido para o PC1.

Tabela 4.2 Protocolo do sector de fabrico

Direcção	Mensagem	Significado
→	LOAD|TURN	Solicita o carregamento do torno
←	LOAD_TURN|WAIT	Em execução
←	LOAD_TURN|ACK	Carregamento efectuado
→	LOAD|MILL	Solicita o carregamento da fresadora
←	LOAD_MILL|WAIT	Em execução
←	LOAD_MILL|ACK	Carregamento efectuado
→	LOAD|TURN_BUFFER	Solicita o carregamento do *buffer* com matérias-primas para o torno
←	LOAD_TURN_BUFFER|WAIT	Em execução
←	LOAD_TURN_BUFFER|ACK	Carregamento efectuado
→	LOAD|MILL_BUFFER	Solicita o carregamento do *buffer* com matérias-primas para a fresadora
←	LOAD_MILL_BUFFER|WAIT	Em execução
←	LOAD_MILL_BUFFER|ACK	Carregamento efectuado
→	UNLOAD|TURN	Solicita o descarregamento do torno
←	UNLOAD_TURN|WAIT	Em execução
←	UNLOAD_TURN|ACK	Descarregamento efectuado
→	UNLOAD|MILL	Solicita o descarregamento da fresadora
←	UNLOAD_MILL|WAIT	Em execução
←	UNLOAD_MILL|ACK	Descarregamento efectuado
→	UNLOAD_TURN_BUFFER	Solicita o descarregamento do *buffer* de produtos acabados do torno
←	UNLOAD_TURN_BUFFER|WAIT	Em execução
←	UNLOAD_TURN_BUFFER|ACK	Descarregamento efectuado
→	UNLOAD_MILL_BUFFER	Solicita o descarregamento do *buffer* de produtos acabados da fresadora
←	UNLOAD_MILL_BUFFER|WAIT	Em execução
←	UNLOAD_MILL_BUFFER|ACK	Descarregamento efectuado
→	START|TURN	Executa o programa NC do torno
←	START_TURN|WAIT	Em execução
←	START_TURN|ACK	Execução iniciada
←	TURN_PROGRAM|COMPLETED	Programa NC concluído no torno
←	TURN_PROGRAM|ERROR	Erro no Programa NC
→	START|MILL	Executa o programa NC da fresadora
←	START_MILL|WAIT	Em execução
←	START_MILL|ACK	Execução iniciada
←	MILL_PROGRAM|COMPLETED	Programa NC concluído na fresadora
←	MILL_PROGRAM|ERROR	Erro no Programa NC
→	LOAD_TURN_PROGRAM|"number"	Carrega o programa "number" no torno
←	LOAD_TURN_PROGRAM|WAIT	Em execução
←	LOAD_TURN_PROGRAM|ACK	Programa NC carregado
→	LOAD_MILL_PROGRAM|"number"	Carrega o programa "number" na fresadora
←	LOAD_MILL_PROGRAM|WAIT	Em execução
←	LOAD_MILL_PROGRAM|ACK	Programa NC carregado
←	"comando"|BUSY	Sector de fabrico ocupado com outra tarefa
←	"comando"|EMPTY	Impossível executar comando porque o *buffer* está vazio de matéria-primas
←	"comando"|FULL	Impossível executar comando porque o *buffer* está cheio de produtos acabados

4.3.2 Sector de montagem

O sector de montagem de peças é constituído por um robô *Scorbot ER VII* [Eshed Robotec, 1996], um *conveyor* e uma mesa de montagem, como mostra a figura 4.6. O objectivo deste

sector é fazer a montagem e embalagem de algumas peças produzidas no sector de fabrico. O controlo, monitorização e coordenação deste sector são da responsabilidade do PC2. Para tal, foi desenvolvido para este sector o software "Assembly Sector", recorrendo ao "winRS232ROBOTcontrol" desenvolvido e apresentada no capítulo 3 [Ferrolho e Crisóstomo, 2004a] e [Ferrolho e Crisóstomo, 2007a].

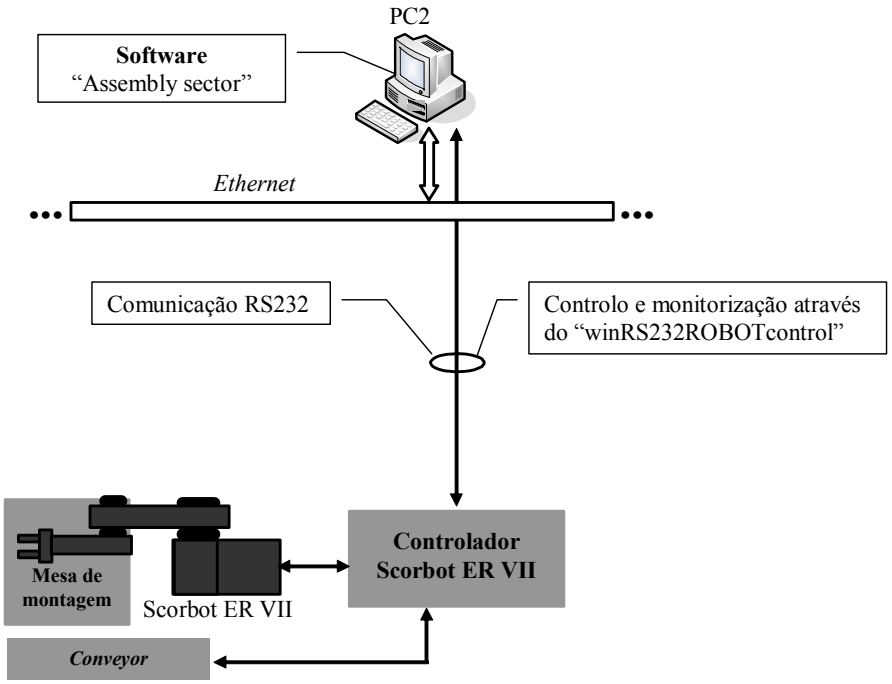


Figura 4.6 Sector de montagem.

4.3.3 Sector de transporte

O sector de transporte é constituído por um transportador no centro da CFF, que interliga os sectores de armazenamento, fabrico e montagem de peças. No transportador, circulam paletes com matérias-primas e produtos acabados. A posição das paletes no transportador é obtida através de um conjunto de sensores indutivos. A paragem das paletes nos vários sectores é conseguida através de *stoppers*, estrategicamente posicionados ao longo do tapete.

O controlo do sector de transporte é feito através do PC3 e de um kit USB (hardware), como mostra a figura 4.7. O kit USB adquire os sinais provenientes dos sensores indutivos e envia-os para o PC3 via porta USB. Quando é necessário activar os *stoppers*, o PC3 envia os sinais de actuação para o kit através da porta USB. A comunicação com o kit USB é conseguida através de um conjunto de funções disponíveis numa DLL [Ferrolho e Crisóstomo, 2006b] e [Ferrolho e Crisóstomo, 2007a].

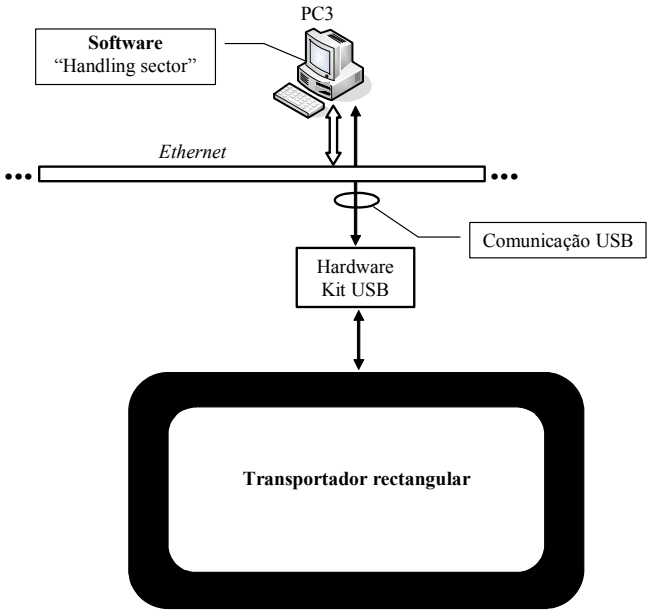


Figura 4.7 Sector de transporte.

A figura 4.8 apresenta o software "Handling Sector" utilizado no controlo e monitorização do sector de transporte. Este software está instalado no PC3 e tem incorporado um servidor de forma a permitir o controlo e a monitorização remota do sector de transporte através da rede *Ethernet*. Desta forma, podemos parar a palete que desejarmos no sector que pretendermos.

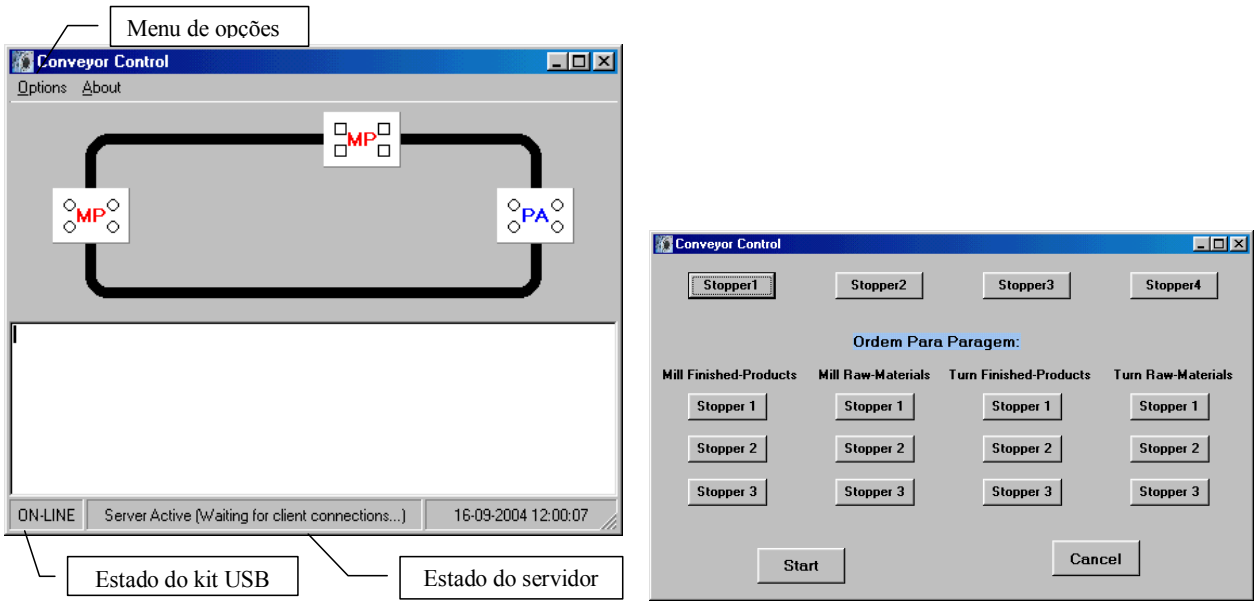


Figura 4.8 Software desenvolvido para o PC3.

Como já foi referido, o software que controla o sector de transporte possui um servidor que permite o controlo e a monitorização remota através da rede *Ethernet*. A tabela 4.3 apresenta o protocolo de comunicação utilizado neste sector.

Tabela 4.3 Protocolo do sector de transporte

Mensagem	Significado
STOP_PALETE|x,y	Pára palete 'x' no *stopper* 'y'
STOP_PALETE|WAIT	À espera da palete
STOP_PALETE|X,Y	Palete 'x' parada no *stopper* 'y'
START_PALETE|_	Baixa todos os *stoppers*
RUNNING|_	Todas as paletes em movimento
STOPPER|y	Baixa *stopper* 'y'
STOPPER_Y|ACK	*Stopper* 'y' em baixo
"comando"|UNKNOWN	Comando desconhecido

4.3.4 Sector de armazenamento

Como mostra a figura 4.9, o sector de armazenamento é constituído por vários armazéns de matérias-primas e produtos acabados, e ainda, por um robô ABB IRB1400 [ABB, 2003e]. Este sector é controlado e monitorizado pelo software "Storage Sector" instalado no PC4, cujas funções são [Ferrolho e Crisóstomo, 2006b] e [Ferrolho e Crisóstomo, 2007a]:

– Administração da base de dados dos vários armazéns;

– Controlar e monitorizar o robô ABB IRB1400;

– Controlar e monitorizar todas as operações que decorrem ao nível deste sector.

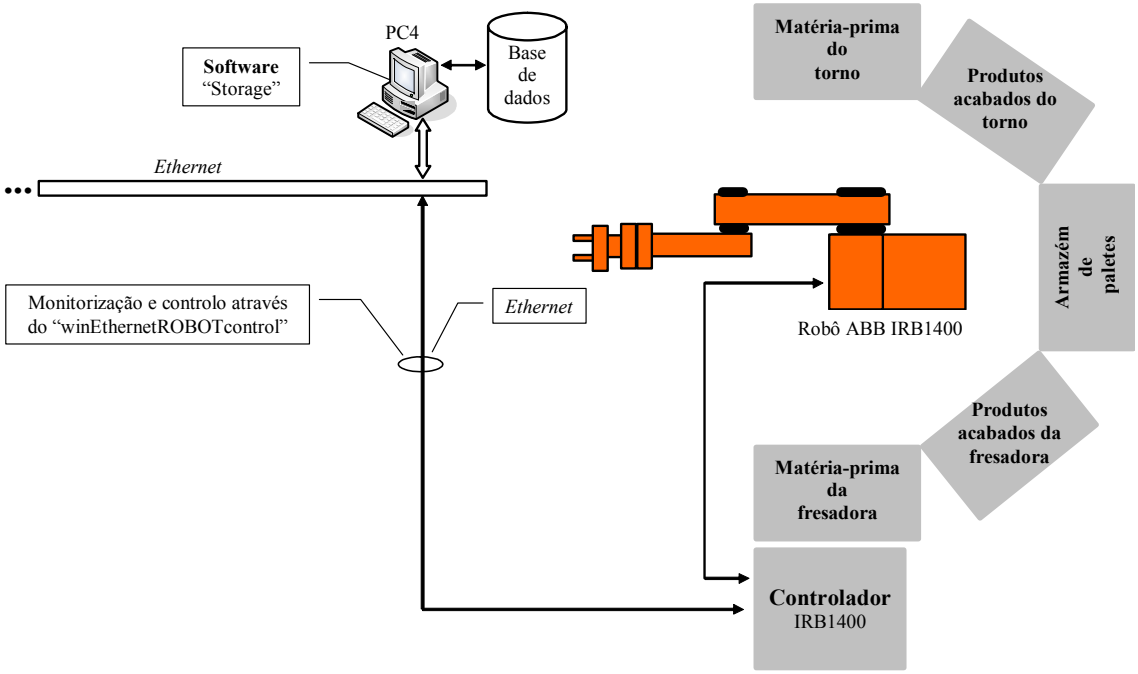


Figura 4.9 Sector de armazenamento.

O controlo e a monitorização do robô ABB IRB1400 são efectuados através das funções, procedimentos e eventos existentes em "winEthernetROBOTcontrol" (ver a tabela 3.2 do capítulo 3). Sempre que há uma palete cheia de produtos acabados a circular no transportador, esta é retirada pelo robô ABB IRB1400 para os armazéns de produtos acabados e substituída por

uma nova palete. Sempre que exista uma palete de matéria-prima vazia a circular no transportador, esta é recarregada pelo robô ABB IRB1400.

A base de dados pode ser actualizada a qualquer momento pelo utilizador, mas também é actualizada automaticamente sempre que o robô ABB IRB1400 efectua uma operação de carregamento, descarregamento ou substituição de paletes. Como apresenta a figura 4.10, o utilizador pode aceder à base de dados através do PC4 e do computador central (gestor da CFF). Em qualquer um destes computadores, o utilizador pode visualizar e editar a informação referente à base de dados do sector de armazenamento. As posições das matérias-primas, produtos acabados e paletes nos vários armazéns são enviadas através de variáveis para os programas que se encontram no controlador do robô ABB IRB1400. Estas variáveis informam o robô onde se encontra a matéria-prima, produtos acabados e paletes nos armazéns.

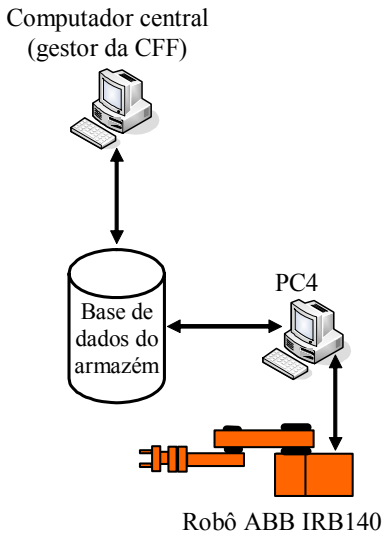


Figura 4.10 Base de dados do sector de armazenamento.

A figura 4.11 mostra o software "Storage Sector", desenvolvido para controlar e monitorizar o sector de armazenamento. Este software está instalado no PC4 e tem incorporado um servidor com o objectivo de permitir o controlo e a monitorização remota deste sector através da rede *Ethernet*. Sucintamente, este software permite: visualizar e actualizar a base de dados dos armazéns, repor as matérias-primas nas paletes, descarregar as paletes de produtos acabados, e substituir as paletes no transportador.

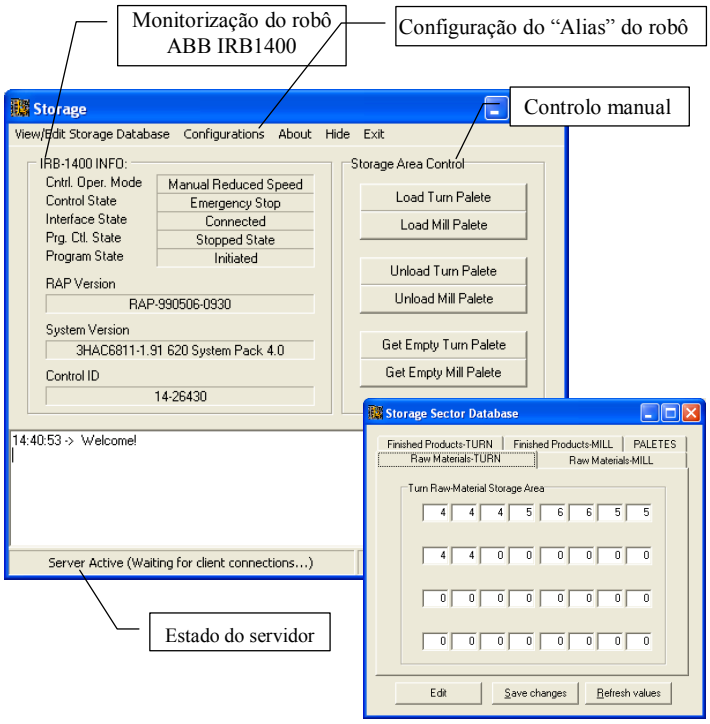


Figura 4.11 Software desenvolvido para o PC4.

A tabela 4.4 apresenta o protocolo de comunicação desenvolvido para o sector de armazenamento. Nesta tabela, o símbolo → significa o envio de uma mensagem do PC cliente (PC4) para o PC servidor (PC do robô ABB IRB1400) e o símbolo ← significa o envio de uma mensagem do PC servidor para um PC cliente (*feedback* do robô).

Tabela 4.4 Protocolo do sector de armazenamento

Direcção	Mensagem	Significado
→	LOAD_PALETE|TURN	Carregamento da palete de MP do torno
←	LOAD_PALETE_TURN|WAIT	Em execução
←	LOAD_PALETE_TURN|ACK	Carregamento efectuado
→	LOAD_PALETE|MILL	Carregamento da palete de MP da fresadora
←	LOAD_PALETE_MILL|WAIT	Em execução
←	LOAD_PALETE_MILL|ACK	Carregamento efectuado
→	UNLOAD_PALETE|TURN	Descarregamento da palete de PA do torno
←	UNLOAD_PALETE_TURN|WAIT	Em execução
←	UNLOAD_PALETE_TURN|ACK	Descarregamento efectuado
→	UNLOAD_PALETE|MILL	Descarregamento da palete de PA da fresadora
←	UNLOAD_PALETE_MILL|WAIT	Em execução
←	UNLOAD_PALETE_MILL|ACK	Descarregamento efectuado
→	GET_PALETE|TURN	Substituição da palete de PA do torno
←	GET_PALETE_TURN|WAIT	Em execução
←	GET_PALETE_TURN|ACK	Substituição efectuada
→	GET_PALETE|MILL	Substituição da palete de PA da fresadora
←	GET_PALETE_MILL|WAIT	Em execução
←	GET_PALETE_MILL|ACK	Substituição efectuado
←	"comando"|BUSY	Sector de armazenamento ocupado
←	"comando"|EMPTY	Armazém vazio
←	"comando"|FULL	Armazém cheio

4.4 Computador central

O computador central (gestor da CFF) controla toda a produção da CFF, interligando os diversos computadores e redes de comunicação de dados, de forma a permitirem o controlo e supervisão em tempo real das operações, recolhendo e processando os fluxos de informação dos vários recursos. Neste computador estão implementados os três primeiros níveis da estrutura hierárquica apresentada na figura 4.2: engenharia, planeamento e sequenciamento. A figura 4.12 apresenta a janela principal do software desenvolvido para o computador central. Genericamente, o computador central é responsável por:

- Desenvolver e desenhar novos produtos a fabricar – nível da engenharia;
- Elaborar planos de produção, montagens e testes dos produtos a serem fabricados – nível do planeamento;
- Encontrar a sequência óptima de processamento das peças, de modo a optimizar a utilização das máquinas CNC – nível do sequenciamento;
- Manter uma base de dados de peças a maquinar, incluindo os respectivos programas NC;
- Sincronizar os vários sectores, de forma a produzir lotes variáveis de diferentes tipos de peças, consoante as encomendas dos clientes;
- Monitorizar o estado actual da produção;
- Garantir tolerância a falhas, segurança e coerência dos dados.

Todos os sectores desenvolvidos para a CFF têm implementado um sistema de segurança, e o computador central possui o sistema principal de segurança. Por exemplo, o PC1 possui um sistema de segurança para o sector da produção, o PC2 apresenta um sistema de segurança para o sector de montagem de peças, etc.. Os sistemas de segurança instalados em cada um dos computadores (PC1, PC2, PC3 e PC4) são responsáveis pela segurança e tolerância a falhas nos respectivos sectores. Optámos por desenvolver um sistema de segurança para cada sector porque os sectores são independentes, e se acontecer algum problema num sector, os restantes não precisam de parar completamente. Por exemplo, se uma das máquinas do sector de fabrico avariar, o PC1 necessita de o saber imediatamente (controlo de tempo real muito rápido), uma vez que o robô ABB IRB140 não poderá efectuar o carregamento ou o descarregamento dessa máquina. O sistema de segurança do PC1 comunica a avaria da máquina ao sistema principal de segurança do computador central (esta comunicação não necessita de um controlo de tempo real muito rápido), e este activa um alarme que chama o operador para repor a normalidade no sector de fabrico. Numa situação destas, o sector de fabrico não necessita de parar completamente, pois

a outra máquina pode continuar a trabalhar. Este exemplo, mostra como a tolerância a falhas e as questões de segurança foram abordadas nos vários sectores da CFF.

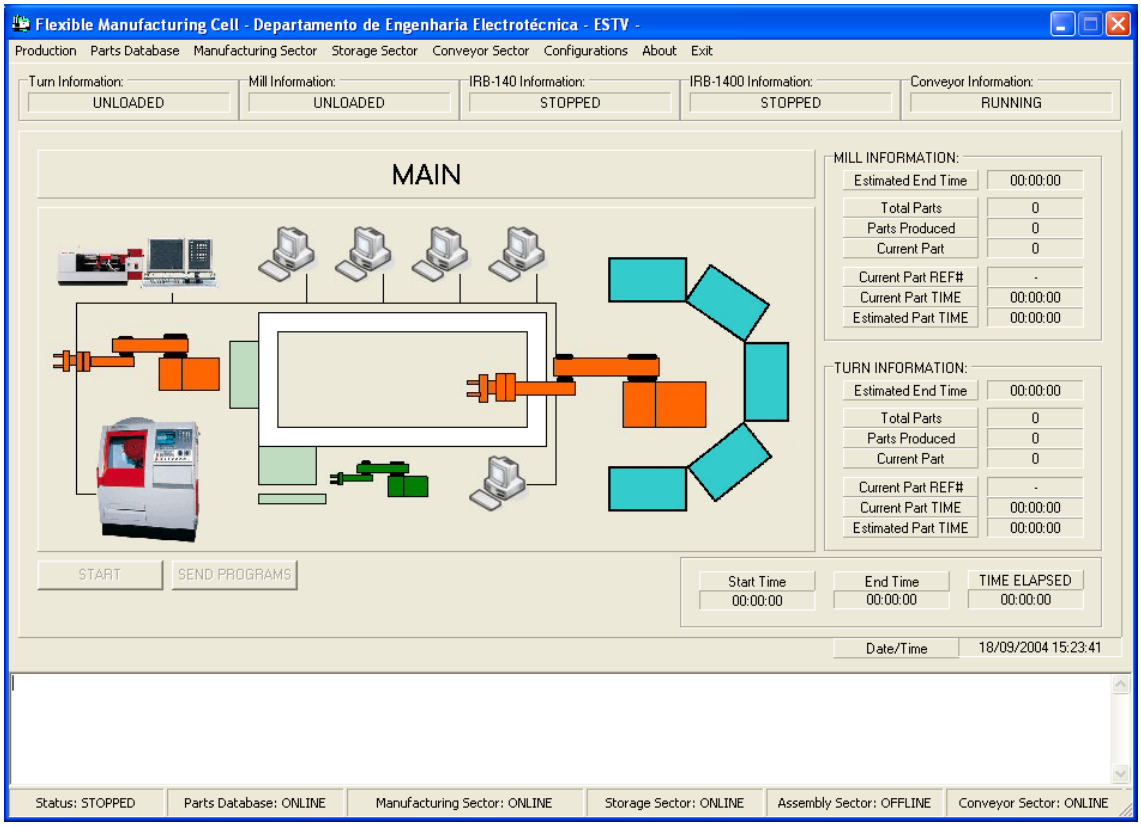
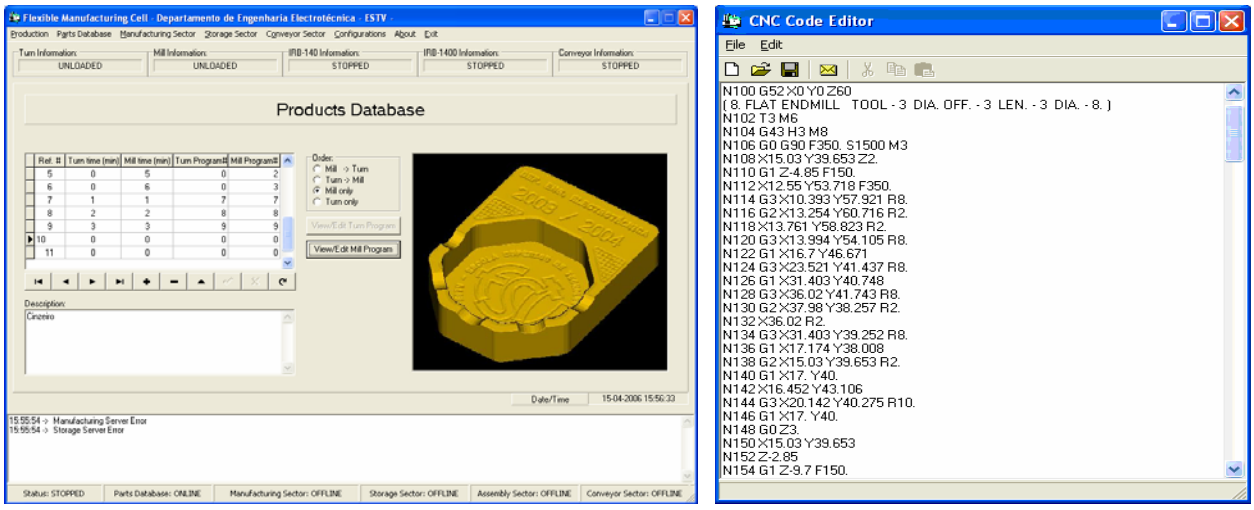


Figura 4.12 Janela principal do software desenvolvido para o computador central.

O primeiro nível da estrutura hierárquica da CFF contém as funções de engenharia e desenho, onde os produtos são desenvolvidos e desenhados com o recurso a software CAD/CAM (*e.g.*, MasterCam). As saídas deste nível são os desenhos e os materiais a serem usados no processo de fabrico.

O planeamento da produção, montagem e testes são feitos no segundo nível da estrutura hierárquica da CFF – planeamento. Umas das saídas deste nível são os programas NC para as máquinas CNC, obtidos através de software CAD/CAM. A base de dados dos produtos (*jobs*) existente no computador central (ver figura 4.3) contém os desenhos para cada *job*, planos de produção, programas NC, entre outras coisas, como mostra a figura 4.13(a). Sempre que é desenvolvido um novo *job*, é necessário colocar toda a informação referente a esse *job* na base de dados. A figura 4.13(b) apresenta parte do programa NC do *job* seleccionado da figura 4.13(a). Se este *job* for seleccionado para ser produzido na CFF, o computador central precisa de enviar o programa NC do *job* em questão, para a máquina CNC.



a) b)

Figura 4.13 Base de dados de produtos e editor de programas NC.

4.4.1 Processo CAD/CAM/CNC

A figura 4.14 mostra o processo CAD/CAM/CNC inerente ao sistema de produção da CFF. O CAD/CAM decorre ao nível do computador central (gestor da CFF). O CAD permite a criação de geometrias e o CAM permite a simulação da maquinagem dos produtos. Na fase de concepção do produto, o sistema CAD/CAM reduz o tempo de projecto e aumenta a rapidez de transferência de informação para o sector de fabrico [Ferrolho, 2001], [Ferrolho e Crisóstomo, 2004b] e [Ferrolho e Crisóstomo, 2004c].

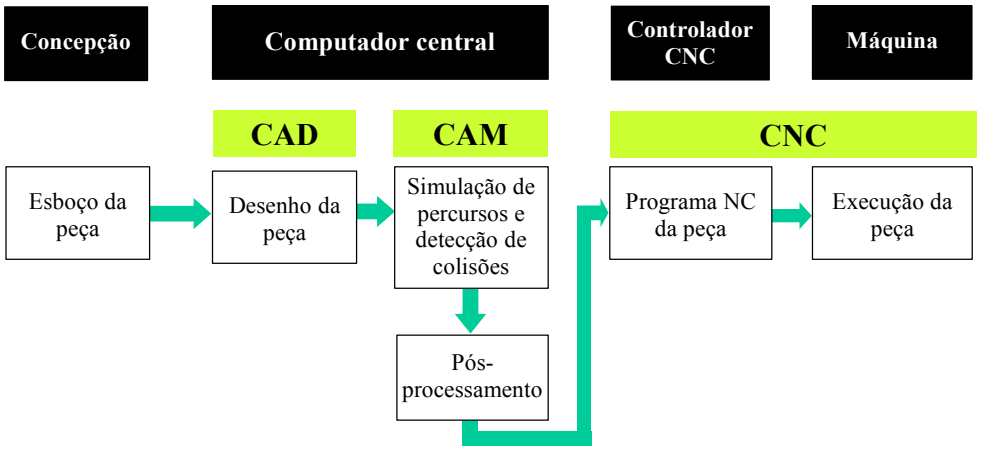


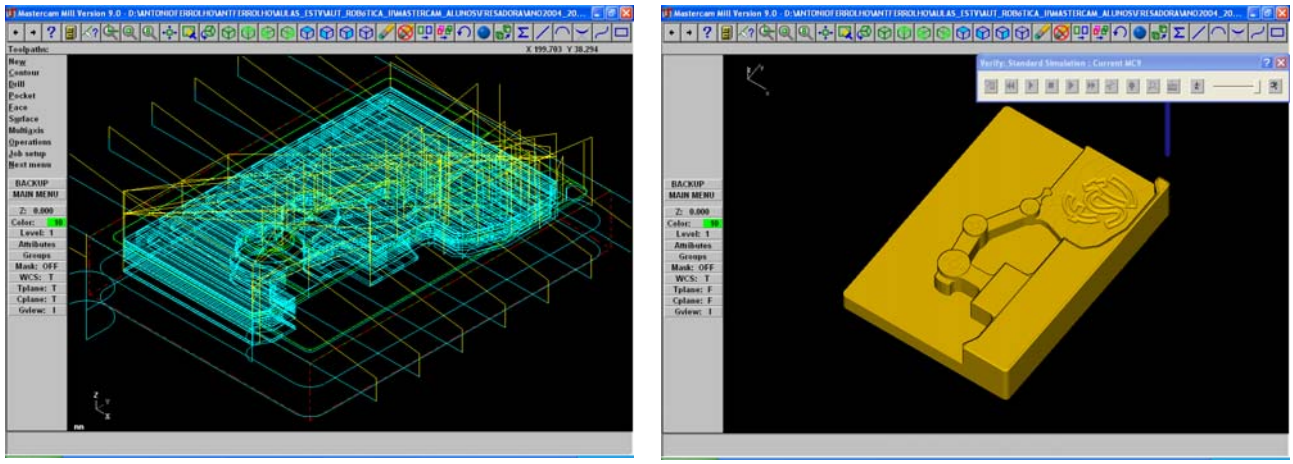
Figura 4.14 Processo CAD/CAM/CNC.

Optámos por utilizar o software *MasterCam* por ser um pacote de software que integra a tecnologia CAD/CAM e opera em ambiente *Windows*. A sua versatilidade e capacidade permitem produzir rapidamente e com precisão peças de alta qualidade. O módulo de CAD disponibiliza uma extensiva capacidade de criação de geometrias a 2D/3D: pontos, rectas, arcos,

superfícies, letras, sólidos, entre outros. O módulo de CAM permite gerar percursos de maquinagem para qualquer máquina de comando numérico: fresagem 2D, 3D; torneamento, entre outros. O uso do computador central (gestor da CFF) no desenho de geometrias e na geração de programas NC possibilita a realização imediata de alterações e a verificação do resultado. O CAD/CAM reduz significativamente o tempo, recursos, e custos de produção com a sua eficiência e precisão.

O objectivo principal de um sistema CAD/CAM, além da modelação de geometrias, é o de gerar programas NC. Um programa NC, mesmo quando é gerado num sistema de CAD/CAM, pode conter erros que se vão encontrar posteriormente, e de forma bastante onerosa, no processo de fabrico, causando prejuízos consideráveis e atrasos no planeamento de fabrico. A fim de evitar tais acontecimentos indesejáveis, torna-se necessário testar os programas, antes destes serem enviados para as máquinas CNC, com o objectivo de detectar possíveis anomalias. A peça, que antes foi modelada no sistema de CAD, tem que estar em conformidade com a peça posteriormente maquinada. Para efectuar estes testes, o *MasterCam* utiliza métodos gráficos computacionais para simulação do processo de remoção do material. Esta informação gráfica é traduzida pela representação da ferramenta, do seu percurso e das formas que vão sendo geradas. Com a simulação, consegue-se reduzir consideravelmente erros de programação, através da detecção de colisões, avanços e rotações das ferramentas mal programadas, entre outros.

A cor azul apresentada na figura 4.15(a) mostra a simulação dos percursos das ferramentas usadas na maquinagem, e a figura 4.15(b) mostra o aspecto final da peça maquinada na fresadora CNC.



a) b)

Figura 4.15 Simulação e aspecto final de uma peça maquinada na fresadora CNC.

Como mostra a figura 4.14, o pós-processador actua na fase final do processo CAD/CAM criando um ficheiro com o programa NC, que permitirá à máquina CNC executar a maquinagem da peça. O ficheiro resultante do pós-processamento é guardado na base de dados dos produtos (ver figura 4.13) juntamente com a restante informação relativa à peça em questão. Este ficheiro, sempre que necessário, é transferido para a máquina CNC através da rede *Ethernet*.

4.4.2 Sequenciamento

Como mostram as figuras 4.2 e 4.3, o sequenciamento está implementado no terceiro nível da estrutura hierárquica da CFF. A flexibilidade da CFF permite um grande número de configurações das estações de trabalho, bem como um conjunto de alternativas para a sequência de operações nas máquinas, tornando o sequenciamento e o controlo muito complexos. Nestes tipos de sistemas de fabrico, os problemas de sequenciamento são muito complexos, e por conseguinte não se pode estabelecer o óptimo como objectivo.

Admitindo que são conhecidos os produtos a fabricar, a sua sequência de fabrico e os recursos disponíveis para o fazer, a tarefa do sequenciamento consiste em determinar a sequência de passagem dos produtos pelos recursos, bem como definir a alocação dos recursos, a partir da definição dos tempos de início e fim das operações, de modo a que um ou mais critérios de desempenho sejam optimizados. Esta descrição está particularizada para o caso dos problemas de sequenciamento em sistemas de produção, como é o caso da CFF. De facto, problemas de sequenciamento surgem associados a áreas muito diferentes de actividade, tais como: a definição de horários numa escola, a definição da ordem de aterragem de aviões num aeroporto, a escolha da ordem de execução de diferentes programas num computador, entre outras. Suponha-se que existem *n jobs* $\{J_1,...,J_n\}$ a serem processados em *m* máquinas $\{M_1,...,M_m\}$. O espaço das soluções possíveis é determinado por $(n!)^m$ [French, 1982]. O processamento de um *job* numa máquina é designado por operação e caracterizado pelo respectivo tempo de processamento p_{ij} (*job i*, máquina *j*). Para cada *job* definem-se as suas restrições tecnológicas, isto é a sequência de operações necessárias ao processamento do *job*. Assim, o problema de sequenciamento consiste em determinar uma sequência de passagem dos *jobs* pelas máquinas de modo que seja: primeiro, compatível com as suas restrições tecnológicas; e segundo, óptima segundo um determinado critério de desempenho.

Foram utilizados algoritmos genéticos (AG) na resolução de problemas de sequenciamento na CFF. Alguns dos problemas de sequenciamento são de muito difícil resolução, mas se o AG for

desenvolvido correctamente este pode obter boas soluções, podendo mesmo obter a solução óptima. Os problemas de sequenciamento são classificados através de quatro parâmetros: *n*/*m*/*A*/*B* [Conway *et al.*, 1967]. O parâmetro *n* representa o número de *jobs*, *m* o número de maquinas, *A* descreve o tipo de problema (*e.g.*, *F* problema *flow-shop*, *P* problema *flow-shop* de permutação e *G* problema *job-shop*), e *B* descreve o critério de desempenho a utilizar. Quando o *m*=1 o parâmetro *A* é deixado em branco [Ferrolho e Crisóstomo, 2005b] e [Ferrolho e Crisóstomo, 2005c].

Foram desenvolvidos, testados e validados algoritmos de sequenciamento para estes tipos de sistemas flexíveis de fabrico. Para isso, desenvolvemos uma ferramenta de software a que chamamos *hybrid and flexible genetic algorithm* (HybFlexGA) [Ferrolho *et al.*, 2006b] e [Ferrolho e Crisóstomo, 2007b]. Os resultados deste estudo são apresentados e largamente discutidos nos capítulos seguintes.

4.5 Resultados experimentais

Esta secção pretende mostrar como a CFF funciona num ambiente real de produção. Utilizamos um exemplo simples (8 *jobs*) para explicarmos melhor os resultados e o funcionamento da CFF [Ferrolho e Crisóstomo, 2007a].

Suponhamos que temos oito *jobs* para serem produzidos na CFF e que o problema de sequenciamento em questão é do tipo *flow-shop*. A base de dados dos produtos contém vários *jobs* disponíveis para produção na CFF, mas o conjunto de oito *jobs* escolhidos para produção foram $\{J_{21}, J_{22}, J_{23}, J_{24}, J_{25}, J_{26}, J_{27}, J_{28}\}$. A tabela 4.5 e a figura 4.16(a) apresentam os tempos de processamento dos *jobs* na fresadora e no torno. O problema de sequenciamento é do tipo $8/2/F/F_{max}$, e o número total de sequências para este exemplo é dado por $(8!)^2$.

O desencadeamento de uma ordem de produção na CFF está dividido em duas fases. Na primeira fase, é necessário proceder às seguintes operações no computador central da CFF:

- Se necessário, visualizar a informação relativa aos *jobs* (*e.g.*, tempo de processamento, desenhos, programas NC, entre outras coisas) e actualizar a base de dados dos *jobs* (ver figura 4.13 e figura 4.16(a));
- Seleccionar o tipo de sequenciamento (*e.g.*, máquina única, *flow-shop*, etc.), os *jobs* a sequenciar, e as respectivas quantidades a maquinar de cada *job*. No exemplo em questão

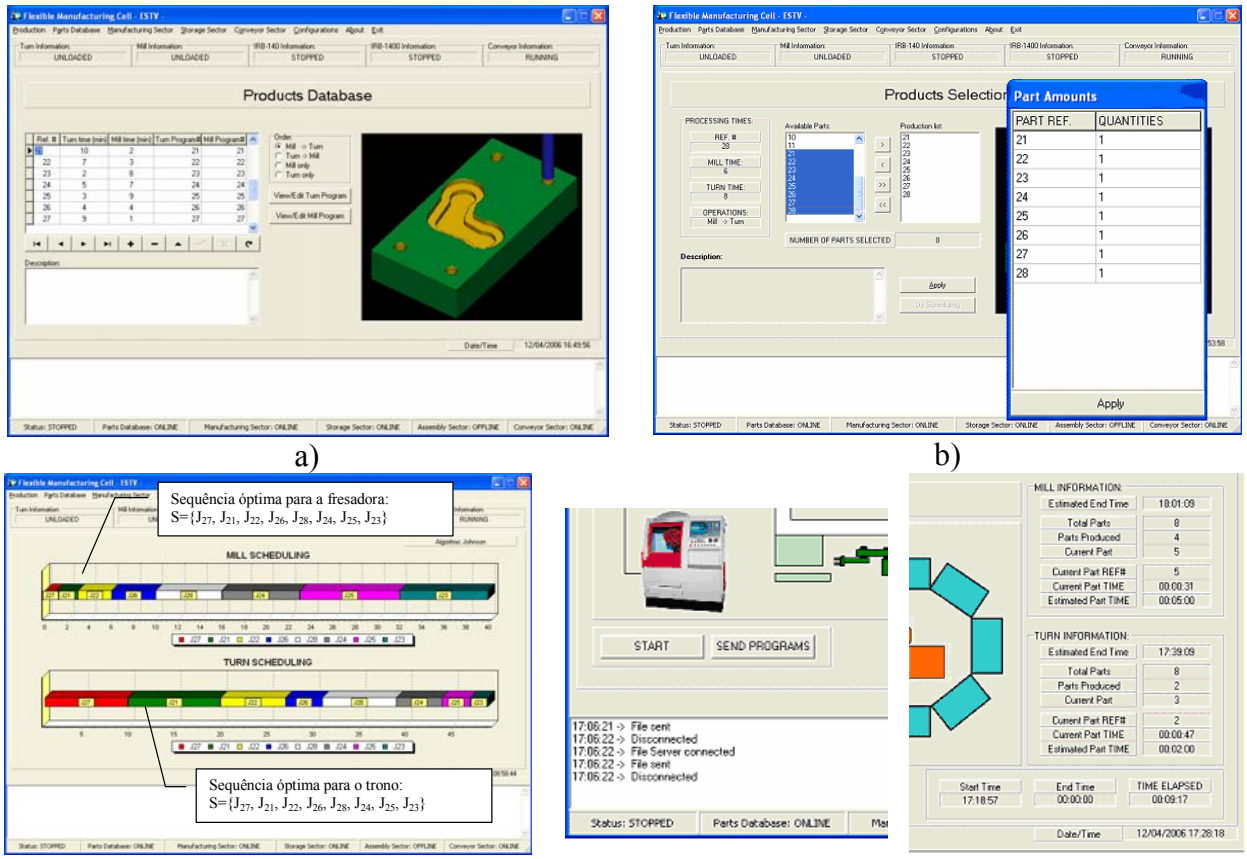
seleccionamos: sequenciamento *flow-shop*, oito *jobs* (J_{21}, J_{22}, J_{23}, J_{24}, J_{25}, J_{26}, J_{27}, J_{28}) e uma unidade para cada *job* (ver figura 4.16(b));

- Obter a sequência óptima e o diagrama de *Gantt* dos *jobs* seleccionados. Para isso, devemos clicar em primeiro lugar no botão "Apply" e de seguida no botão "Do scheduling" (ver figura 4.16(b)). Depois de aplicado o algoritmo genético (HybFlexGA) ao problema 8/2/F/F_{max}, foi obtido o diagrama de *Gantt* apresentado na figura 4.16(c), cuja a sequência é S={J_{27}, J_{21}, J_{22}, J_{26}, J_{28}, J_{24}, J_{25}, J_{23}} com F_{max}=49;
- Transferir os programas NC dos *jobs* seleccionados para as máquinas CNC através do botão "SEND PROGRAMS" (ver figura 4.16(d)). Os programas NC transferidos para as máquinas CNC contêm toda a informação necessária para a maquinagem dos mesmos, como por exemplo: tipo e quantidades de ferramentas a usar, tipos de trajectórias a usar pelas ferramentas, velocidades de avanço, tipo de acabamento, entre outras;
- Utilizar o botão "START" para dar ordem à CFF para iniciar a produção (ver figura 4.16(d)).

Na segunda fase, depois de se ter clicado no botão "START", a CFF inicia a produção dos *jobs* de acordo com o diagrama de *Gantt* obtido (ver a figura 4.16(c)). Durante a fase de produção dos *jobs*, o computador central da CFF (gestor da CFF) coordena e supervisiona todos os sectores da CFF (sector de fabrico, sector de montagem de peças, sector de transporte, e sector de armazenamento). O computador central da CFF é informado em tempo real do estado actual da produção, como por exemplo: o número de *jobs* produzidos nas máquinas (ver na figura 4.16(e) "Parts produced"), o tempo estimado para o final da produção (ver na figura 4.16(e) "Estimated End Time"), o tempo de produção decorrido (ver na figura 4.16(e) "ELAPSED TIME"), a quantidade de peças a produzir nas máquinas (ver na figura 4.16(e) "Total Parts"), entre outras.

Tabela 4.5 Tempos de processamento nas máquinas

Job	**Fresadora**	**Torno**
21	2	10
22	3	7
23	8	2
24	7	5
25	9	3
26	4	4
27	1	9
28	6	8



a) b) c) d) e)

Figura 4.16 Sequência das operações na CFF.

4.6 Resumo do Capítulo

No âmbito deste capítulo foi desenvolvida uma CFF com o intuito de testarmos em aplicações industriais as potencialidades das ferramentas de software desenvolvidas e apresentadas no capítulo 3. A CFF é constituída por quatro sectores, sendo o controlo e monitorização dos equipamentos de cada sector realizado por quatro computadores (PC1, PC2, PC3 e PC4). A coordenação, sincronização e integração dos quatro sectores é feita pelo computador central "gestor da CFF". Neste capítulo, foram ainda apresentados o *layout* e a estrutura hierárquica da CFF. A estrutura hierárquica da CFF é baseada em cinco níveis: engenharia, planeamento, sequenciamento, controlo e aquisição de dados. A interligação de todos os equipamentos na CFF (quer de fabrico quer de processamento de informação) é feita através de uma rede de computadores *Ethernet*.

Como ficou demonstrado neste capítulo, as ferramentas de software desenvolvidas no capítulo 3 foram fundamentais na resolução dos problemas de coordenação, integração e controlo de todos os equipamentos utilizados na CFF. Os resultados práticos obtidos na CFF demonstram a

viabilidade e o sucesso do software desenvolvido na coordenação dos diferentes sectores utilizados na CFF e no melhoramento das suas performances. Sucintamente, as ferramentas de software utilizadas na CFF permitem:

- O desenvolvimento de software de controlo e monitorização, para os vários sectores da CFF, utilizando uma única linguagem de programação (*e.g.*, C++);
- O controlo e monitorização em tempo real das operações na CFF;
- A gestão da base de dados dos *jobs*, incluindo os respectivos programas NC;
- A monitorização constante da produção;
- A tolerância a falhas, segurança e coerência de dados;
- A recuperação de situações de paragem dos robôs, colocando os robôs em posições de segurança;
- A transferência e a execução de programas nos robôs e máquinas CNC;
- A coordenação, integração e sincronização de diferentes equipamentos provenientes de diferentes fabricantes na CFF;
- A aplicação de algoritmos de sequenciamento para optimização das máquinas CNC.

Por último, os resultados experimentais obtidos neste capítulo mostram, também, a viabilidade e o sucesso das ferramentas de software desenvolvidas na coordenação dos diferentes sectores na CFF, bem como no melhoramento das suas performances.

Capítulo 5

Sequenciamento de Tarefas na Célula Flexível de Fabrico

"Subitamente, o mundo tornou-se tão rápido que nem há tempo para pensar bem nas decisões complexas a tomar."
Bill Gates, 2001, Executive Digest, p. 22

5.1 Introdução

A flexibilidade das Células Flexíveis de Fabrico (CFF) permite um grande número de configurações das estações de trabalho, bem como um conjunto de alternativas para a sequência de operações nas máquinas, tornando o planeamento, o sequenciamento e o controlo muito complexos. Os problemas de sequenciamento (*scheduling*) em CFF são muito complexos, porque há muitas variáveis em jogo. A capacidade do sistema, as avarias nas máquinas e a falta de matérias-primas são algumas dessas variáveis. Visto ser um problema complexo nem sempre se pode estabelecer o óptimo como objectivo.

Neste capítulo é abordado o problema de sequenciamento de tarefas (*jobs*) em Células Flexíveis de Fabrico. Inicialmente, apresenta-se o problema do sequenciamento, terminologia, conceitos utilizados ao longo desta tese e um exemplo motivador de sequenciamento. São, ainda, abordados os problemas de sequenciamento numa máquina, em duas máquinas e em três máquinas. Por último, é feito o estudo computacional destes problemas de sequenciamento, apresentando-se, também, a arquitectura do software desenvolvido e implementado na CFF, bem como os testes computacionais realizados.

5.2 O Problema do Sequenciamento

O planeamento da produção envolve frequentemente a resolução de problemas de sequenciamento com um impacto importante no desempenho das organizações. Tais problemas consistem basicamente em, dado um conjunto limitado de recursos, genericamente designados por máquinas, determinar a sua utilização ao longo do tempo, de modo a processar um conjunto

de tarefas (*jobs*) independentes ou interrelacionadas, com vista à satisfação de objectivos de natureza económica e operacional. Assim, o sequenciamento consiste em determinar a sequência de passagem das tarefas pelos recursos, bem como definir a alocação dos recursos a partir da definição dos tempos de início e fim das operações, de modo a que um ou mais critérios de desempenho sejam optimizados. Os problemas de sequenciamento surgem associados a áreas muito diferentes de actividade, como por exemplo, definição de horários numa escola, ordem de aterragem de aviões num aeroporto, alocação de tarefas, entre outros.

Os problemas de sequenciamento têm sido tradicionalmente abordados como problemas de optimização sujeitos a restrições, cujos elementos básicos são as máquinas e as tarefas. Em [Baker, 1974] é referido que os problemas de sequenciamento podem ser abordados em quatro fases principais: a formulação, a análise, a síntese e a avaliação. Na formulação é identificado o tipo de problema e determinado o critério ou critérios que orientarão o processo de decisão. A análise consiste no processo de examinar detalhadamente os elementos do problema e as suas inter-relações. Nesta fase são identificadas as variáveis de decisão, inter-relações entre elas e as restrições a que devem obedecer. Na fase de síntese são construídas soluções alternativas admissíveis, caracterizando as diferentes linhas de acção disponíveis. A avaliação é o processo de comparação dessas alternativas e de selecção de uma solução para o problema, com base nos critérios desenvolvidos na fase de formulação do problema.

Os problemas reais de sequenciamento são de resolução muito complexa, sendo conhecidos em termos de teoria da complexidade como NP difíceis (*NP-hard*), tendo dificuldade não polinomial, para os quais o tempo requerido para a determinação de um plano óptimo ou sub--óptimo aumenta exponencialmente com o tamanho do problema [Morton *et al.*, 1993].

O sequenciamento visa a afectação no tempo de recursos escassos designados por máquinas, a actividades ou tarefas (*jobs*), sujeitos às restrições básicas de que em qualquer instante nenhuma máquina processa mais do que uma tarefa, e nenhuma tarefa é processada em simultâneo, em mais do que uma máquina. Como textos de referência sobre sequenciamento, temos [Conway *et al.*, 1967], [Baker, 1974], [French, 1982], [Morton *et al.*, 1993], [Sule, 1997], [Blazewicz *et al.*, 2001] e [Brucker, 2001], entre outros.

Tradicionalmente, os métodos de sequenciamento foram desenvolvidos tendo como referência a resolução de problemas de natureza industrial, no fabrico de bens. Por esse facto, tornou-se

corrente a utilização de vocabulário associado à produção industrial [Conway *et al.*, 1967], [Baker, 1974], [French, 1982] e [Madureira, 2003]. No entanto, a sua aplicação pode ser igualmente considerada para a área de prestação de serviços, como por exemplo, em problemas de tratamento de doentes num hospital, na reparação de automóveis numa garagem, ou na organização e controlo de tráfego aéreo. Os recursos e as tarefas podem assumir diferentes formas, conforme o ambiente em que estão inseridos. Assim, os recursos podem ser máquinas num ambiente fabril, pistas em aeroportos, médicos, enfermeiros e salas em hospitais, unidades de processamento em computadores. Por tarefas pode entender-se os processos de fabrico de produtos num ambiente fabril, descolagens e aterragens de aviões em aeroportos, tratamento de doentes em hospitais, execução de programas em computadores, para além de outros. As tarefas a sequenciar são caracterizadas por diferentes parâmetros, tipicamente o tempo de processamento, a data de entrega e a data de lançamento. Os objectivos podem ser distintos, tais como, a minimização do tempo de conclusão ou do número de tarefas em atraso e a maximização da utilização dos recursos.

Na literatura encontram-se várias definições para sequenciamento. Por exemplo, Baker em [Baker, 1974] define sequenciamento da seguinte forma:

> "O *sequenciamento é a atribuição de recursos no tempo para o processamento de um conjunto de tarefas*".

French [French, 1982] define sequenciamento da seguinte maneira:

> "O *problema de sequenciamento consiste em encontrar uma sequência de passagem das tarefas pelos recursos, correspondendo a um plano exequível e óptimo em relação a um qualquer critério de optimização adoptado*".

Blazewicz, Ecker, Pesch, Smith e Weglarz [Blazewicz *et al.*, 2001] definem sequenciamento da seguinte forma:

> "O*s problemas de sequenciamento podem ser definidos duma forma geral como um problema de atribuição de recursos ao longo do tempo para a realização de um conjunto de tarefas*".

Outras definições podem ser encontradas na literatura, como por exemplo em Conway [Conway *et al.*, 1967] e Artiba [Arbita *et al.*, 1997], entre outras. No entanto, as definições apresentadas acima são as que consideramos de maior interesse prático.

5.3 Sequenciamento: terminologia e conceitos

Nesta secção apresentamos a terminologia e alguns conceitos usados ao longo desta tese. A terminologia de sequenciamento deriva da indústria de processamento e manufactura. Fala-se, por isso, em tarefas (j*obs*) e máquinas (*Machines*), embora os objectos em questão, em alguns casos, não tenham qualquer semelhança com tarefas ou com máquinas.

Uma tarefa (ou *Job)* é composta por um conjunto de operações que podem ser sequenciais, ou concorrentes, necessárias para a produção de um produto ou lotes de produtos. Sucintamente, podemos definir *job* como sendo um conjunto de operações que permitem obter um produto final. Ao longo desta tese, no que diz respeito ao problema de sequenciamento, os termos tarefa e *job* serão usados indistintamente. Uma máquina pode ser caracterizada através das suas capacidades funcionais ou qualitativas (operações que é capaz de efectuar) e quantitativas (taxa de produção, tempo de processamento, tempos de preparação, etc.).

Nos problemas de sequenciamento supõe-se que existem n tarefas $\{J_1, J_2, J_3,..., J_n\}$ que são processadas em m Máquinas $\{M_1, M_2, M_3,..., M_m\}$. Cada tarefa é processada uma única vez por cada máquina. O processamento de uma tarefa por uma máquina é designado por *Operação*. A operação O_{ij} representa o processamento da tarefa i pela máquina j. As *Restrições Tecnológicas* especificam a ordem pela qual cada tarefa tem de ser processada nas máquinas. A ordem de processamento de uma tarefa é independente das restantes tarefas. Chama-se *Tempo de Processamento* p_{ij} ao tempo de execução da operação O_{ij}. Geralmente, assume-se que esse tempo é constante e conhecido antecipadamente. Normalmente, considera-se que p_{ij} engloba o tempo necessário para o transporte de uma tarefa e de preparação da máquina para o processamento de O_{ij}. Assume-se, também, que uma máquina que não esteja a processar, esteja sempre disponível. O instante em que a tarefa J_i chega ou fica disponível para o início do processamento, designa-se por *Instante de Chegada* (*Ready Time*) *ri* da tarefa.

A seguir, apresenta-se um resumo da notação utilizada neste e nos próximos capítulos. A figura 5.1 ajuda a compreender melhor a notação usada.

J_i – *job* i

M_j – máquina j

O_{ij} – operação do *job* i na máquina j

p_{ij} – tempo de processamento da operação *Oij*

r_i – instante de chegada (*ready time*) do *job* i

d_i – data de entrega (*due date*) do *job* i

a_i – período disponível (*allowance*) para o processamento do *job* i ($a_i=d_i-r_i$)

W_{ik} – tempo de espera (*waiting time*) do *job* i antes da operação *k*

W_i – tempo total de espera *(total waiting time*) do *job* i ($W_i = \sum_{k=1}^{m} W_{ik}$)

C_i – instante de finalização (*completion time*) do *job* i ($C_i = r_i + \sum_{k=1}^{m} \left[W_{ik} + p_{ij(k)} \right]$)

F_i – tempo de fluxo (*flow time*) do *job* i ($F_i=C_i-r_i$)

L_i – demora (*lateness*) do *job* i ($L_i=C_i-d_i$)

T_i – atraso (*tardiness*) do *job* i ($T_i=\max\{0, L_i\}$

E_i – antecipação (*earliness*) do *job* i ($E_i=\max\{-L_i, 0\}$

I_j – tempo de inactividade (*idle time*) da máquina j ($I_j = C_{\max} - \sum_{i=1}^{n} p_{ij}$)

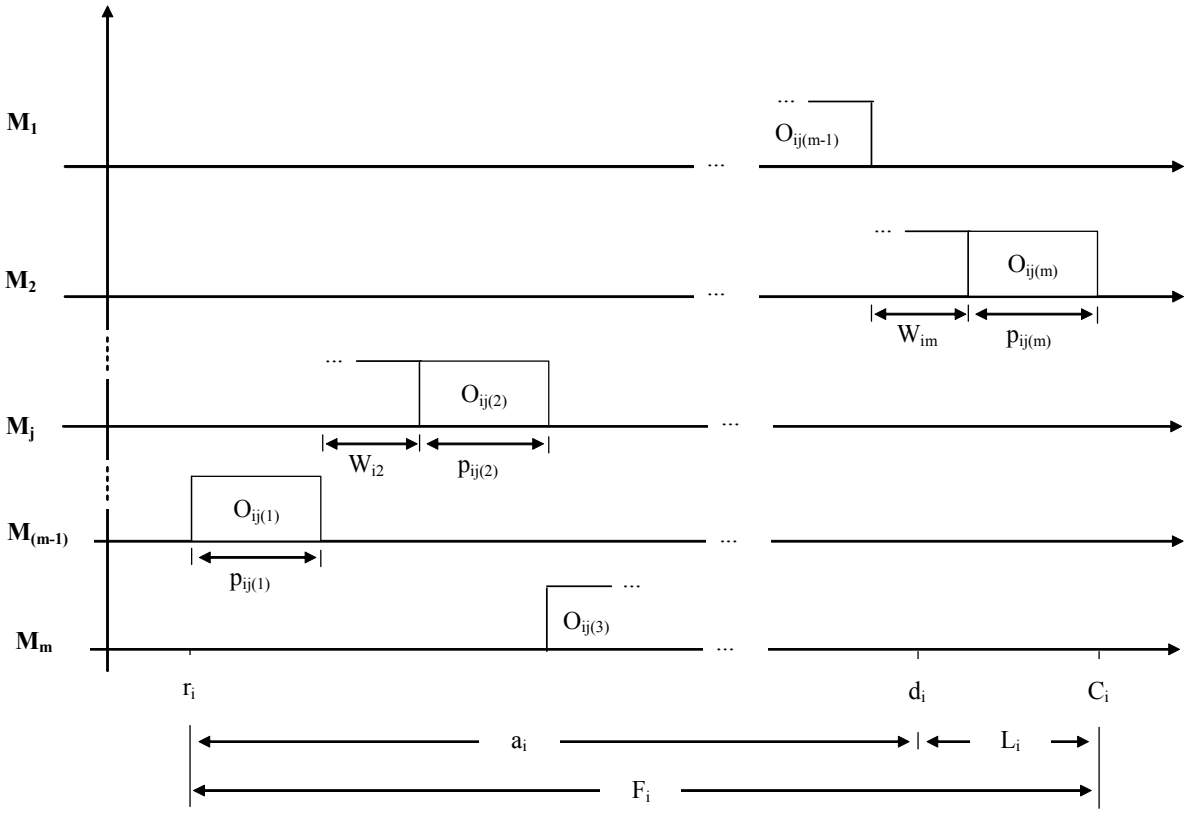


Figura 5.1 Diagrama de *Gantt* para o *job* J_i.

Com o objectivo de facilitar a identificação de problemas de sequenciamento, e por consequência os métodos para a sua resolução, alguns autores propuseram nomenclaturas para a representação deste tipo de problemas. Um dos métodos de classificação encontrados na literatura é a notação com quatro campos – *n*/*m*/*A*/*B* – descrito em [Conway *et al.*, 1967], onde:

n - número de *Jobs*

m – número de máquinas

A – tipo de problema:

- *F* – problema *Flow-Shop*
- *P* – problema *Flow-Shop* de permutação (todas as máquinas processam os *jobs* pela mesma ordem)
- *G* – problema *Job-Shop*

B – Critério de desempenho:

- Maximizar o ritmo de saída.
- Satisfazer os requisitos do cliente em termos de qualidade e pontualidade.
- Minimizar os custos de utilização, baseados em *stocks* e/ou inactividade das máquinas.

Para o exemplo $n/2/F/C_{max}$ temos: *n jobs*, 2 máquinas, problema do tipo *Flow-Shop* e o critério de desempenho é minimizar o C_{max}.

Para qualquer problema de sequenciamento o número de sequências possíveis é obtido por $(n!)^m$. Para um problema de sequenciamento, existirá sempre uma sequência óptima? A resposta a esta pergunta é afirmativa, porque o espaço de soluções é finito. Mas, nem sempre se pode estabelecer o óptimo como objectivo.

Das sequências possíveis, $(n!)^m$, estas podem ser divididas em sequências factíveis e não-factíveis. Nas sequências factíveis encontram-se os sequenciamentos semi-activos, como mostra a figura 5.2.

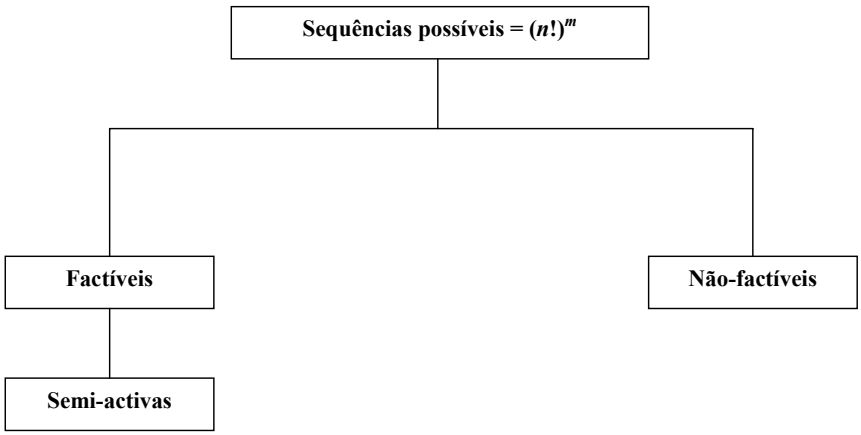


Figura 5.2 Sequências factíveis, não-factíveis e semi-activas.

O sequenciamento factível (*feasible*) é compatível com as restrições tecnológicas e o sequenciamento não factível (*infeasible*) não é compatível com as restrições tecnológicas. Um sequenciamento é semi-activo se não existir nenhuma operação que possa começar mais cedo sem alterar a sequência e sem violar as restrições tecnológicas. As restrições tecnológicas são as sequências de operações necessárias ao processamento de um *job*. Por exemplo, suponhamos que são necessárias as seguintes operações para o processamento dum *job*: 1º maquinagem da peça, 2º limpeza da peça, 3º controlo de qualidade. Para este exemplo, o sequenciamento factível é: 1-2-3. Como exemplo de sequenciamentos não factíveis, temos: 2-1-3; 3-2-1; 1-3-2; entre outros.

O objectivo do sequenciamento é determinar uma sequência de passagem dos *jobs* pelas máquinas que seja compatível com as restrições tecnológicas e que seja óptima segundo um determinado critério de desempenho. As restrições tecnológicas estão relacionadas com a ordem de processamento das operações de cada *job*. A seguir, apresentam-se os critérios de desempenho mais utilizados:

1. Maximizar o ritmo de saída, baseado nos tempos de finalização:
   - $F_{\max}$, minimizar o tempo de fluxo máximo (*flow time*);
   - $C_{\max}$, minimizar o tempo total de finalização da produção (*completion time*);
   - $\overline{F}$, média do tempo de fluxo (*mean flow-time*);
   - $\overline{C}$, média do instante de finalização (*mean completion time*).
2. Satisfazer os requisitos do cliente, baseados nas datas de entrega:
   - $L_{\max}$, a máxima demora *(maximum lateness*);
   - $T_{\max}$, o máximo atraso (*maximum tardiness*);
   - $\overline{L}$, média de demora do *job* i (*mean lateness*);
   - $\overline{T}$, média de atraso do *job* i (*mean tardines*).
3. Minimizar os custos de utilização, baseados em *stocks* ou na inactividade das máquinas.
   - $N_{w(t)}$, número de *jobs* em espera (*waiting*) ou por chegar no instante t;
   - $N_{p(t)}$, número de *jobs* em processamento (*processing*) no instante t;
   - $N_{c(t)}$, número de *jobs* terminados (*completed*) até ao instante t;
   - $N_{u(t)}$, número de *jobs* por terminar (*uncompleted*) no instante t;
   - $I_{\max}$, máximo tempo de inactividade (*idle*) da máquina.
4. Também podem ser utilizados outros critérios de desempenho, tais como:
   - Tempo de fluxo ponderado: $F_{wt} = \sum_i w_i F_i$
   - Demora ponderada: $L_{wt} = \sum_i w_i L_i$

- Atraso ponderado: $T_{wt} = \sum_i w_i T_i$

Em qualquer problema de sequenciamento temos:

– O número total de *jobs* (n): $n=N_{w(t)}+N_{p(t)}+N_{c(t)}$

– O número de *jobs* por terminar no instante t: $N_{u(t)} = N_{w(t)}+N_{p(t)}$

– O número de *jobs* por terminar no instante t=0: $N_{u(t)} = n$

– O número de *jobs* por terminar no instante $t=C_{\max}$: $N_{u(C\max)} = 0$

– A média dos *jobs* por terminar no instante t: $\overline{N}_u = \frac{1}{C_{\max}} \int_0^{C_{\max}} N_{u(t)} dt$

Clarificando alguns conceitos, tem-se:

- ***Restrições Tecnológicas*** – Ordem de processamento das operações de cada tarefa.
- ***Sequência*** – Ordem de processamento das tarefas em cada máquina.
- ***Alocação Temporal*** – Definição dos tempos de início e finalização das operações.
- ***Sequenciamento*** – Sequência + Alocação Temporal.

Resumindo, resolver um problema de sequenciamento consiste em descobrir uma sequência de processamento das tarefas em cada máquina que seja:

- Exequível, isto é, compatível com as restrições tecnológicas;
- Óptima, de acordo com algum critério de desempenho ou função objectivo.

O critério de desempenho mais utilizado é o de minimizar o *Tempo Total de Finalização da Produção*, denominado por ***Cmax*** (***makespan***).

Se considerarmos a variabilidade da informação, podemos classificar o sequenciamento em duas categorias [Baker, 1974], [French, 1982] e [Sule, 1997]:

- Estáticos, se todas as tarefas a tratar estiverem disponíveis no início do período de sequenciamento e não sofrerem modificações ao longo do tempo. Pressupõem normalmente que o conjunto de tarefas a programar chegam em simultâneo e que esse conjunto não varia até o processamento estar concluído.
- Dinâmicos, se o número de tarefas variar ao longo do tempo.

Quanto à variabilidade de parâmetros, os problemas de sequenciamento podem ser classificados em:

- Determinísticos, se todos os parâmetros das tarefas são conhecidos à *priori*. Assim, se existirem dados de lançamento diferentes para cada tarefa, estes são conhecidos.
- Não-determinísticos[1], correspondendo a um cenário real de fabrico sujeito a factores aleatórios de mudança que provocam alterações no estado do sistema [Blazewicz *et al.*, 1994]. Assim, o plano de sequenciamento necessita de ser dinamicamente adaptado (reescalonado) sempre que eventos inesperados ocorrem no sistema, isto é, se o número de tarefas variar ao longo do tempo, se as máquinas avariarem, etc.

Por definição, consideramos que os problemas estáticos são implicitamente determinísticos, dado que todos os parâmetros do problema são previamente conhecidos no início do período de sequenciamento. Assim, a classificação quanto à variabilidade de parâmetros apenas se justifica quando se está na presença de problemas dinâmicos. A generalidade dos sistemas reais de fabrico são dinâmicos não-determinísticos, ou seja, o seu estado é alterado pela possibilidade de ocorrência contínua de eventos aleatórios.

5.4 Exemplo motivador

Suponha que quatro amigos vivem juntos num apartamento. Todos eles têm o hábito de ler quatro jornais pela manhã antes de saírem de casa. A ordem, os tempos de leitura dos jornais e a hora a que cada um se levanta encontram-se na tabela 5.1.

Tabela 5.1 Ordem e tempos de leitura dos jornais

Leitor	Levanta-se	Ordem e tempo de leitura			
Alberto	8:30	Expresso (60)	Semanário (30)	DN (2)	CM (5)
Beatriz	8:45	Semanário (75)	DN (3)	Expresso (25)	CM (10)
Carlos	8:45	DN (5)	Semanário (15)	Expresso (10)	CM (30)
Diana	9:30	CM (90)	Expresso (1)	Semanário (1)	DN (1)

A pergunta que fazemos é: a que horas podem os quatro amigos sair juntos? A resposta a esta pergunta não é imediata, porque estamos perante um problema de sequenciamento. Para podermos responder a esta pergunta, temos primeiro que resolver o problema de sequenciamento. A tabela 5.2 apresenta uma possível solução para este problema de sequenciamento, e a figura 5.3 o respectivo diagrama de *Gantt*.

[1] Alguns autores designam esta classe de problemas como estocásticos.

Tabela 5.2 Ordem de leitura dos quatro jornais

	Ordem de leitura			
Jornal	**1º**	**2º**	**3º**	**4º**
Expresso	**A**lberto	**D**iana	**C**arlos	**B**eatriz
Semanário	**B**eatriz	**C**arlos	**A**lberto	**D**iana
DN	**C**arlos	**B**eatriz	**A**lberto	**D**iana
CM	**D**iana	**A**lberto	**C**arlos	**B**eatriz

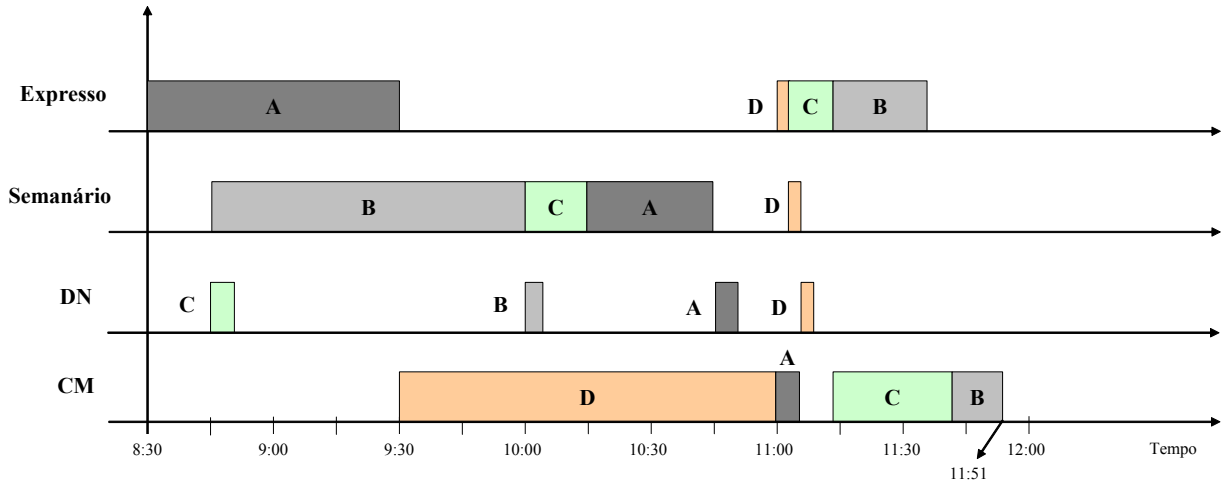


Figura 5.3 Diagrama de *Gantt*.

Para o exemplo apresentado, podemos considerar a seguinte analogia:

– Leitores → *Jobs*

– Jornais → Máquinas

– Leitura de um jornal → Operação

– Tabela de leituras → Restrições tecnológicas

– Hora de saída → Objectivo

Para o exemplo motivador apresentado temos 331.776 sequências possíveis, resultantes de 4 leitores (n = 4) e 4 jornais (m = 4). Se tivéssemos um software num computador que nos analisasse 1000 sequências por segundo, teríamos que esperar 5,53 minutos para que fossem analisadas todas as sequências possíveis. Se acrescentássemos mais 1 leitor ao problema (equivalente a mais 1 *job*), teríamos de esperar 57,6 horas para que o software analisasse as $2{,}0736 \times 10^8$ sequências possíveis, uma vez que $(5!)^4 = 2{,}0736 \times 10^8$. A resolução do problema de sequenciamento tem evidentemente imensas soluções. A pergunta que se coloca é: como escolher uma?

5.5 Complexidade computacional

Em termos de dificuldade de resolução, os problemas de optimização combinatória, como os problemas de sequenciamento em sistemas de produção são, em geral, classificados de acordo com a complexidade computacional. De facto, existe uma quantidade significativa de bibliografia que contém resultados relativamente à complexidade dos problemas de sequenciamento. Sínteses desses resultados podem ser consultadas em [Baker, 1974], [Garey *et al.*, 1979], [Lawler, 1983], [Blazewicz *et al.*, 1994], [Blazewicz *et al.*, 2001], [Brucker, 2001], entre outros.

Em Optimização Combinatória é habitual falar-se em problemas fáceis e problemas difíceis. Para os primeiros, também conhecidos como problemas da classe polinomial P, existe sempre um algoritmo eficiente para a sua resolução. Para os segundos, não é conhecido qualquer algoritmo determinístico eficiente que os resolva em tempo útil sendo, por isso, classificados na classe NP [Garey *et al.*, 1979]. Designam-se por NP-difíceis (*NP-hard*) os problemas para os quais não são conhecidos algoritmos eficientes de resolução, devido à sua complexidade exponencial.

Existe outra classe de problemas correspondente aos problemas de maior dificuldade na classe NP. Os problemas dessa classe são denominados de NP-completos (*NP-complete*). São, por vezes, conhecidos como problemas "*fáceis de definir e difíceis de resolver*". Muitos dos problemas de optimização combinatória são intratáveis, pertencendo à classe dos problemas NP--completos (*Non-deterministic polynomial-time complete*). Um algoritmo óptimo para a resolução de um problema nesta classe poderá requerer um número de passos computacionais que cresce exponencialmente com a dimensão do problema. Assim, só os problemas de pequena dimensão podem ser resolvidos.

Por eficácia consideramos a obtenção de soluções de boa qualidade, e por eficiência a obtenção de soluções em tempo útil. Um algoritmo é considerado eficiente se a complexidade temporal crescer polinomialmente e não exponencialmente com a dimensão do problema. Para um algoritmo eficiente, o aumento do poder de cálculo dos computadores tem efeito multiplicativo na dimensão dos problemas que podem ser abordados. Pelo contrário, para os algoritmos de complexidade exponencial o efeito de aumento de cálculo computacional é somente aditivo [French, 1982]. Assim, as dificuldades existentes na resolução deste tipo de problemas não serão resolvidos apenas com o aparecimento de computadores mais rápidos, pelo menos com recurso às tecnologias actualmente vigentes.

5.6 Problemas de sequenciamento com uma só máquina

Para problemas de uma só máquina, as medidas $\overline{C}$, $\overline{F}$, $\overline{W}$, $\overline{L}$, $\overline{N}_u$, $\overline{N}_w$ são equivalentes. A relação de equivalência entre medidas de desempenho mostra que, se uma solução for óptima em relação a uma das medidas, também é óptima relativamente a outra medida. Para problemas de sequenciamento com uma só máquina, as $n!$ permutações dos n *jobs* constituem todas as soluções possíveis [French, 1982] e [Ferrolho e Crisóstomo, 2005c].

5.6.1 Regra SPT (*Shortest Processing Time Scheduling)*

Consideremos a figura 5.4 onde $p_I > p_K$. Depois de analisarmos a figura concluímos que $F_I = a + p_I$, $F_K = a + p_I + p_K$, $F'_I = a + p_k + p_I$ e $F'_K = a + p_K$. Donde se tira que $F_I + F_K = 2a + 2p_I + p_K$ e $F'_I + F'_K = 2a + p_I + 2p_K$.

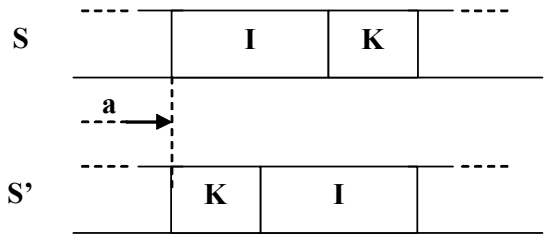


Figura 5.4 Sequenciamento *S* e *S'*.

Considere-se o problema $n/1//\overline{F}$ onde,

$$\overline{F} = \frac{1}{n}\sum_{i=1}^{n} F_i = \frac{1}{n}\sum_{i=1}^{n}(W_i + p_i) = \frac{1}{n}\sum_{k=1}^{n}(W_{i(k)} + p_{i(k)}) = \frac{1}{n}\sum_{k=1}^{n} W_{i(k)} + \frac{1}{n}\sum_{k=1}^{n} p_{i(k)} \tag{5.1}$$

Como $\sum_{k=1}^{n} p_{i(k)} = \sum_{i=1}^{n} p_i$ é constante, pretende-se minimizar $\sum_{k=1}^{n} W_{i(k)}$ logo:

$$\begin{aligned} W_{i(1)} &= 0 \\ W_{i(2)} &= p_{i(1)} \\ W_{i(3)} &= p_{i(1)} + p_{i(2)} \\ &\vdots \\ W_{i(n)} &= p_{i(1)} + \cdots + p_{i(n-1)} \end{aligned} \tag{5.2}$$

Por exemplo, para minimizarmos $W_{i(3)}$ temos que escolher $J_{i(1)}$ e $J_{i(2)}$ com os tempos de processamento mais curtos da lista $\{J_1, J_2, J_3, \ldots, J_n\}$.

Podemos concluir que para o problema $n/1//\overline{F}$ a solução óptima verifica $p_{i(1)} \le p_{i(2)} \le p_{i(3)} \le \cdots \le p_{i(n)}$. Esta solução é conhecida como a regra SPT - *Shortest Processing Time Scheduling*. Esta regra minimiza $\overline{F}$, $\overline{C}$, $\overline{W}$, $\overline{L}$, $\overline{N}_u$ ou $\overline{N}_w$, pois são medidas equivalentes [French, 1982].

Exemplo 5.1 - Aplicação da regra SPT

Considere os valores apresentados na tabela 5.3. Pretendemos determinar a sequência óptima e calcular $\overline{F}$.

Tabela 5.3 Tempos de processamento

J_i	1	2	3	4	5	6	7
p_i	6	4	8	3	2	7	1

Aplicando a regra SPT obtemos a seguinte sequência óptima, $S_{opt}=(J_7, J_5, J_4, J_2, J_1, J_6, J_3)$. Para a sequência óptima encontrada temos,

$$\overline{F} = \frac{1}{n}\sum_{i=1}^{n} F_i = \frac{1}{n}\sum_{i=1}^{n}(W_i + p_i) = \frac{90}{7} \tag{5.3}$$

5.6.2 Regra EDD (*Earliest Due Date Scheduling*)

A regra EDD é apropriada para evitar grandes desvios em relação à data de entrega, como por exemplo, quando todos os *jobs* são necessários noutro processo. Esta regra também é usada para minimizar L_{max} (máxima demora), T_{max} (máximo atraso) e n_T (nº de *jobs* atrasados). Para o problema $n/1//L_{max}$, a sequência óptima satisfaz: $d_{i(1)} \le d_{i(2)} \le d_{i(3)} \le \cdots \le d_{i(n)}$.

Exemplo 5.2 - Aplicação da regra EDD

Considere os tempos de processamento e as datas de entrega apresentadas na tabela 5.4. Pretendemos determinar a sequência óptima para minimizar T_{max} e T_{max}.

Tabela 5.4 Tempos de processamento e datas de entrega

J_i	1	2	3	4	5	6
d_i	7	3	8	12	9	3
p_i	1	1	2	4	1	3

Aplicando a regra EDD, obtemos a seguinte sequência óptima $S_{opt}=(J_6, J_2, J_1, J_3, J_5, J_4)$. Analisando a tabela 5.5 verificamos que o atraso máximo é igual a 1 ($T_{max}=1$).

Tabela 5.5 Instante de finalização, data de entrega, demora e atraso

Job $J_{i(k)}$	**Instante de finalização** $C_{i(k)} = \sum_{l=1}^{k} p_{i(l)}$	**Data de entrega** $d_{i(k)}$	**Demora** $L_{i(k)} = C_{i(k)} - d_{i(k)}$	**Atraso** $T_{i(k)} = \max(0, L_{i(k)})$
6	3	3	0	0
2	4	3	1	1
1	5	7	-2	0
3	7	8	-1	0
5	8	9	-1	0
4	12	12	0	0

5.7 Problemas de sequenciamento com duas máquinas

Nesta secção, iremos abordar os problemas de sequenciamento com duas máquinas. Serão apresentados os problemas *Flow-Shop* e *Job-Shop* e alguns algoritmos de sequenciamento que permitem determinar a sequência óptima para os referidos problemas.

5.7.1 Algoritmo de *Johnson* para o problema $n/2/F/F_{max}$

Nos problemas de *Flow-Shop* de permutação todas as máquinas processam os *jobs* pela mesma ordem. Suponhamos que estamos perante o problema $n/2/F/F_{max}$, isto é: temos que processar n *jobs* em de duas máquinas, cada *job* tem que ser processado pela ordem M_1, M_2, com o objectivo de minimizar o tempo de fluxo máximo (F_{max}).

O algoritmo de *Johnson*, apresentado na figura 5.5, permite determinar a sequência óptima para este tipo de problemas [French, 1982].

1. Seja k=1, l=n (n - nº total de *jobs*)
2. Seja $\{J_1, J_2, ..., J_n\}$ a lista de *jobs* por sequenciar
3. Encontrar o menor a_i ou b_i para todos os *jobs* da lista ($a_i=p_{i1}$ e $b_i=p_{i2}$)
4. Se o menor valor for a_i, então:
   i) Sequenciar J_i na posição k
   ii) Remover J_i da lista
   iii) Incrementar k
   iv) Saltar para o passo 6
5. Se o menor valor for b_i, então:
   i) Sequenciar J_i na posição l
   ii) Remover J_i da lista
   iii) Decrementar l
   iv) Saltar para o passo 6
6. Se existem *jobs* por sequenciar, retornar ao passo 3

Figura 5.5 Algoritmo de *Johnson* para problemas do tipo *Flow-Shop*.

Exemplo 5.3 – Aplicação do algoritmo a um caso prático

A tabela 5.6 mostra os tempos de processamento de 8 *jobs* em duas máquinas (M_1 e M_2). Pretendemos determinar a sequência óptima para o problema $8/2/F/F_{max}$.

Tabela 5.6 Tempos de processamento dos *jobs* nas máquinas

Job	M_1	M_2
1	2	10
2	3	7
3	8	2
4	7	5
5	9	3
6	4	4
7	1	9
8	6	8

A seguir, encontra-se a aplicação do algoritmo de *Johnson* a este problema.

1. k=1 e l=8
2. L={J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8}
3. min{a_i, b_i}=a_7
4. S_{opt}={J_7, __, __, __, __, __, __, __}

   L={J_1, J_2, J_3, J_4, J_5, J_6, J_8}

   k=2
3. min{a_i, b_i}=a_1
4. S_{opt}={J_7, J_1, __, __, __, __, __, __}

   L={J_2, J_3, J_4, J_5, J_6, J_8}

   k=3
3. min{a_i, b_i}=b_3
5. S_{opt}={J_7, J_1, __, __, __, __, __, J_3}

   L={J_2, J_4, J_5, J_6, J_8}

   l=7
3. min{a_i, b_i}=a_2
4. S_{opt}={J_7, J_1, J_2, __, __, __, __, J_3}

   L={J_4, J_5, J_6, J_8}

   k=4
3. min{a_i, b_i}=b_5
5. S_{opt}={J_7, J_1, J_2, __, __, __, J_5, J_3}

   L={J_4, J_6, J_8}

l=6

3. min{a_i, b_i}=a_6
4. S_{opt}={J_7, J_1, J_2, J_6, __, __, J_5, J_3}
   L={J_4, J_8}
   k=5
3. min{a_i, b_i}=b_4
5. S_{opt}={J_7, J_1, J_2, J_6, __, J_4, J_5, J_3}
   L={J_8}
   l=5
3. min{a_i, b_i}=a_8
4. S_{opt}={J_7, J_1, J_2, J_6, J_8, J_4, J_5, J_3}
   L={}
   k=6

Do resultado obtido, podemos concluir que a sequência óptima é S_{opt}={J_7, J_1, J_2, J_6, J_8, J_4, J_5, J_3}. Os primeiros *jobs* apresentam baixos tempos de processamento na máquina 1 e os últimos *jobs* apresentam baixos tempos de processamento na máquina 2 [Ferrolho e Crisóstomo, 2005b].

5.7.2 Algoritmo de *Johnson* para o problema $n/2/G/F_{max}$

Consideremos o seguinte problema do tipo *Job-Shop* $n/2/G/F_{max}$. Neste problema, temos que processar *n jobs* em duas máquinas com o objectivo de minimizar o tempo de fluxo máximo (F_{max}). O algoritmo de *Johnson*, apresentado na figura 5.6, permite determinar a sequência óptima para este tipo de problemas [French, 1982].

Definir 4 classes:

A: *jobs* a processar apenas na máquina M_1
B: *jobs* a processar apenas na máquina M_2
C: *jobs* a processar em ambas as máquinas, na ordem $M_1 \rightarrow M_2$
D: *jobs* a processar em ambas as máquinas, na ordem $M_2 \rightarrow M_1$

Algoritmo:

1. Sequenciar os *jobs* do tipo A por qualquer ordem.
2. Sequenciar os *jobs* do tipo B por qualquer ordem.
3. Sequenciar os *jobs* do tipo C segundo o algoritmo de *Johnson* para o problema $n/2/F/F_{max}$.
4. Sequenciar os *jobs* do tipo D segundo o algoritmo de *Johnson* para o problema $n/2/F/F_{max}$.
5. A sequência óptima será:
   M_1 (S_C, S_A, S_D)
   M_2 (S_D, S_B, S_C)

Figura 5.6 Algoritmo de *Johnson* para problemas do tipo *Job-Shop*.

Exemplo 5.4 – Aplicação do algoritmo a um caso prático

A tabela 5.7 apresenta a ordem e os tempos de processamento de 9 *jobs* em duas máquinas. Pretendemos determinar a sequência óptima para o problema 9/2/G/F_{max}.

Tabela 5.7 Ordem e tempos de processamento

Job	1ª Máquina		2ª Máquina	
1	M_1	8	M_2	2
2	M_1	7	M_2	5
3	M_1	9	M_2	8
4	M_1	4	M_2	7
5	M_2	6	M_1	4
6	M_2	5	M_1	3
7	M_1	9	----	----
8	M_2	1	----	----
9	M_2	5	----	----

A seguir mostra-se a aplicação do algoritmo de *Johnson* a este problema.

Tipo A: J_7

$S_A=\{J_7\}$

Tipo B: J_8, J_9

$S_B=\{J_8, J_9\}$

Tipo C: J_1, J_2, J_3, J_4

Algoritmo de Johnson 4/2/F/Fmax

$S_C = \{\_\_, \_\_, \_\_, J_1\}$

$= \{J_4, \_\_, \_\_, J_1\}$

$= \{J_4, \_\_, J_2, J_1\}$

$= \{J_4, J_3, J_2, J_1\}$

Tipo D: J_5, J_6

Algoritmo de Johnson 2/2/F/Fmax

$S_D = \{\_\_, J_6\}$

$= \{J_5, J_6\}$

Sequência óptima encontrada para a máquina 1 e para a máquina 2:

$S_{opt\_Maq1} = \{J_4, J_3, J_2, J_1, J_7, J_5, J_6\}$

$S_{opt\_Maq2} = \{J_5, J_6, J_8, J_9, J_4, J_3, J_2, J_1\}$

5.8 Problemas de sequenciamento com três máquinas

Nesta secção, iremos abordar os problemas de sequenciamento com três máquinas. Serão apresentados alguns algoritmos de sequenciamento que permitem determinar a sequência óptima para este tipo de problemas.

5.8.1 Algoritmo de *Johnson* para o problema $n/3/F/F_{max}$

O algoritmo de *Johnson* para o problema $n/2/F/F_{max}$ pode ser aplicado a alguns problemas com três máquinas. Para tal é necessário que se verifique uma das seguintes condições:

$$\min_{i=1}^{n}\{p_{i1}\} \geq \max_{i=1}^{n}\{p_{i2}\}$$

$$\text{ou} \qquad (5.4)$$

$$\min_{i=1}^{n}\{p_{i3}\} \geq \max_{i=1}^{n}\{p_{i2}\}$$

Isto é, o menor tempo de processamento da máquina 1 ou máquina 3 deve ser maior ou igual ao maior tempo de processamento da máquina 2. Caso uma destas duas condições se verifique deve-se considerar,

$a_i = p_{i1} + p_{i2}$

$b_i = p_{i2} + p_{i3}$,

sendo a_i e b_i os tempos de processamento em duas máquinas. Desta forma, converte-se um problema de sequenciamento de três máquinas em um de duas máquinas. Depois de calculados a_i e b_i é possível utilizar o algoritmo de *Johnson* para o problema $n/2/F/F_{max}$.

Exemplo 5.5 – Aplicação do algoritmo a um caso prático

A tabela 5.8 apresenta os tempos de processamento em três máquinas. Pretendemos encontrar a sequência óptima para o problema $6/3/F/F_{max}$.

Tabela 5.8 Tempos de processamento em três máquinas

Job	M_1	M_2	M_3
1	10	9	10
2	2	8	10
3	3	7	10

4	6	6	10
5	1	5	10
6	4	4	10

Para os valores apresentados na tabela 5.8 verifica-se a condição $\min_{i=1}^{n}\{p_{i3}\} \geq \max_{i=1}^{n}\{p_{i2}\}$. Assim, podemos utilizar o algoritmo de *Johnson* para o problema $n/2/F/F_{\max}$, desde que façamos $a_i=p_{i1}+p_{i2}$ e $b_i=p_{i2}+p_{i3}$. A tabela 5.9 apresenta os tempos de processamento de M'_1 e M'_2.

Tabela 5.9 Tempos de processamento de M'_1 e M'_2

Job	**M'_1**	**M'_2**
1	19	19
2	10	18
3	10	17
4	12	16
5	6	15
6	8	14

Tendo em conta os resultados da tabela 5.9 e aplicando o algoritmo de *Johnson* para o problema $n/2/F/F_{\max}$, chegamos à seguinte sequência óptima: $S_{opt} = (J_5, J_6, J_2, J_3, J_4, J_1)$.

5.8.2 O método *Branch and Bound*

Se não considerarmos os métodos heurísticos, o método *Branch and Bound* é provavelmente o mais usado em sequenciamento. Este método procura dividir o conjunto de sequenciamentos possíveis em sucessivos sub-conjuntos (mutuamente exclusivos). Em vez de analisarmos os sequenciamentos um a um, vamos considerar sub-conjuntos cujos sequenciamentos possuem características comuns que vão permitir uma análise ao nível de sub-conjuntos, isto é, dividir para conquistar. Estes conjuntos podem ser representados através duma árvore. Os nós da árvore são associados a conjuntos de sequenciamentos. Os sub-conjuntos são caracterizados por um limite inferior da medida de desempenho. Isto significa que devemos procurar identificar um limite inferior para os valores da medida de desempenho de todos os sequenciamentos pertencentes a um dado conjunto (quando estivermos interessados em minimizar o valor da medida de desempenho). São utilizados os limites inferiores da medida de desempenho dos vários conjuntos analisados para concluir que determinados sub-conjuntos podem ser eliminados. Assim, se for encontrado um sequenciamento cuja medida de desempenho é C1, pode-se eliminar o conjunto de sequenciamentos cujo limite inferior é superior a C1.

Para explicarmos melhor o princípio de funcionamento deste método considere:

T – a sequência de *k jobs* sequenciados;

T' – a sequência de (*n-k*) *jobs* por sequenciar

$\alpha_{i(k)}$ – o tempo de finalização do *job* $J_{i(k)}$ na máquina 1;

$\beta_{i(k)}$ – o tempo de finalização do *job* $J_{i(k)}$ na máquina 2;

$\gamma_{i(k)}$ – o tempo de finalização do *job* $J_{i(k)}$ na máquina 3.

A figura 5.7 apresenta os tempos de finalização em três máquinas, sendo $\alpha_{i(k)}$, $\beta_{i(k)}$ e $\gamma_{i(k)}$ os tempos de finalização do *job k* na máquina 1, máquina 2 e máquina 3.

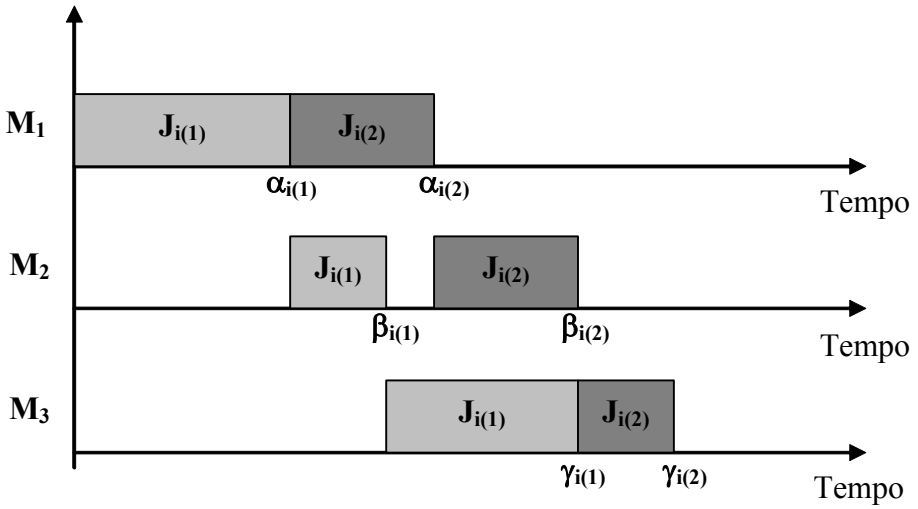


Figura 5.7 Tempos de finalização em três máquinas.

Da figura 5.7 concluímos que:

$$\alpha_{i(1)} = p_{i(1)1}$$
$$\beta_{i(1)} = p_{i(1)1} + p_{i(1)2}$$
$$\gamma_{i(1)} = p_{i(1)} + p_{i(1)2} + p_{i(1)3}$$

$$\begin{aligned}
&\alpha_{i(2)} = p_{i(1)1} + p_{i(2)1} = \alpha_{i(1)} + p_{i(2)1} \\
&\beta_{i(2)} = \max\{\alpha_{i(2)}, \beta_{i(1)}\} + p_{i(2)2} \\
&\gamma_{i(2)} = \max\{\beta_{i(2)}, \gamma_{i(1)}\} + p_{i(2)3} \\
&\vdots \\
&\alpha_{i(k)} = \alpha_{i(k-1)} + p_{i(k)1} \\
&\beta_{i(k)} = \max\{\alpha_{i(k)}, \beta_{i(k-1)}\} + p_{i(k)2} \\
&\gamma_{i(k)} = \max\{\beta_{i(k)}, \gamma_{i(k-1)}\} + p_{i(k)3}
\end{aligned} \tag{5.5}$$

Para determinarmos o *Lower Bound* devemos:

1) Compactar o processamento da máquina M_1

$$C_{\max} = \underbrace{\alpha_{i(k)}}_{T} + \underbrace{\sum_{J_i \in T'} p_{i1}}_{Constante} + \underbrace{p_{i(n)2} + p_{i(n)3}}_{Depende\ de\ T'}$$

$$C_{\max} \geq \alpha_{i(k)} + \sum_{J_i \in T'} p_{i1} + \min_{J_i \in T'}\{p_{i2} + p_{i3}\} \tag{5.6}$$

2) Compactar o processamento da máquina M_2

$$C_{\max} = \beta_{i(k)} + \sum_{J_i \in T'} p_{i2} + p_{i(n)3}$$

$$C_{\max} \geq \beta_{i(k)} + \sum_{J_i \in T'} p_{i2} + \min_{J_i \in T'}\{p_{i3}\} \tag{5.7}$$

3) Compactar processamento da máquina M_3

$$C_{\max} \geq \gamma_{i(k)} + \sum_{J_i \in T'} p_{i3} \tag{5.8}$$

Considerando as expressões 5.6, 5.7 e 5.8, o *Lower Bound* (*lb(T)*) é determinado pela seguinte expressão:

$$lb(T) = \max\{\alpha_{i(k)} + \sum_{J_i \in T'} p_{i1} + \min_{J_i \in T'}\{p_{i2} + p_{i3}\};\ \beta_{i(k)} + \sum_{J_i \in T'} p_{i2} + \min_{J_i \in T'}\{p_{i3}\};\ \gamma_{i(k)} + \sum_{J_i \in T'} p_{i3}\} \tag{5.9}$$

Exemplo 5.6 – Aplicação do método *Branch and Bound* a um caso prático

A tabela 5.10 mostra os tempos de processamento de quatro *jobs* em três máquinas. Pretendemos determinar a sequência óptima para o problema $4/3/F/C_{\max}$.

Tabela 5.10 Tempos de processamento em três máquinas

$\mathbf{J_i}$	1	2	3	4
p_{i1}	3	11	7	10
p_{i2}	4	1	9	12
p_{i3}	10	5	13	2

A figura 5.8 apresenta a árvore com os vários nós para o problema $4/3/F/C_{\max}$. O nó inicial é representado por "XXXX", em que "X" indica que ainda não foi atribuído nenhum *job* a esta posição da sequência. Aos nós "1XXX", "2XXX", "3XXX", "4XXX" foi atribuído um *job* à primeira posição da sequência.

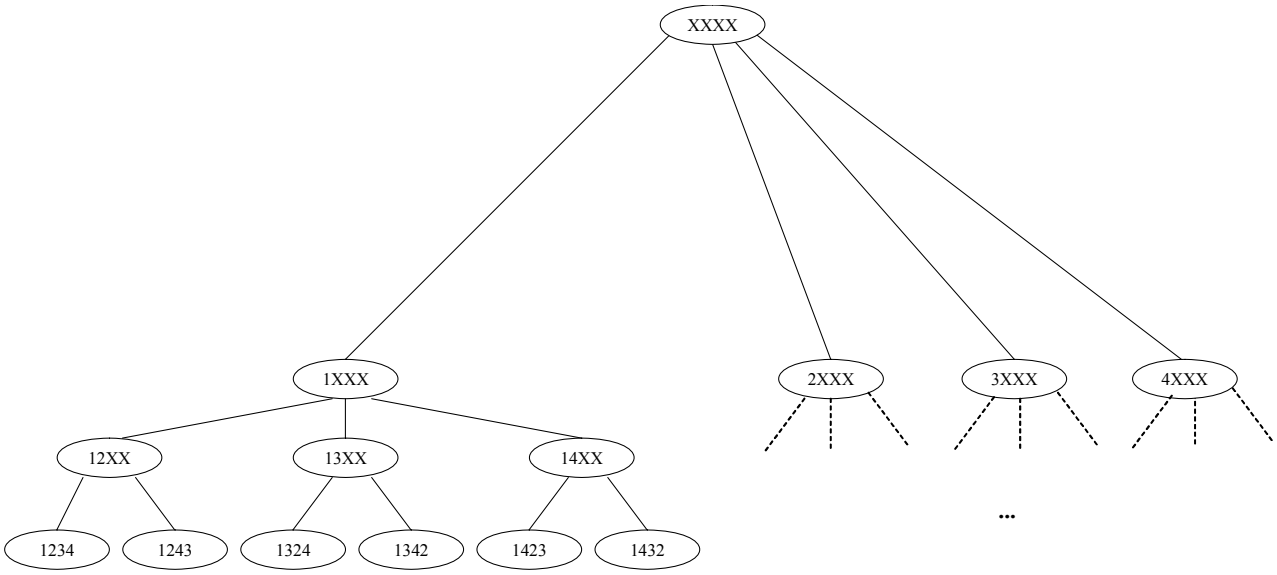


Figura 5.8 Árvore com os nós para o problema 4/3/F/C_{max}.

A estrutura da árvore é utilizada para eliminar os sequenciamentos não óptimos (*the elimination tree*). A seguir, apresentamos o cálculo do *Lower Bound* para alguns dos nós da árvore apresentada na figura 5.8.

Nó 1XXX

$T=\{J_1\}$; $T'=\{J_2, J_3, J_4\}$

$\alpha_1=3$; $\beta_1=7$; $\gamma_1=17$

$lb(T)=\max\{3+28+6, 7+22+2, 17+20\}=37$

Nó 12XX

$T=\{J_1, J_2\}$; $T'=\{J_3, J_4\}$

$\alpha_2=14$; $\beta_2=\max\{14,7\}+1=15$; $\gamma_2=\max\{15, 17\}+5=22$

$lb(T)=\max\{14+17+14, 15+21+2, 22+15\}=45$

Nó 1234

$T=\{J_1, J_2, J_3, J_4\}$; $T'=\{\}$

$\alpha_3=21$; $\beta_3=\max\{21, 15\}+9=30$; $\gamma_3=\max\{30, 22\}+13=43$

$\alpha_4=31$; $\beta_4=\max\{31, 30\}+12=43$; $\gamma_4=\max\{43, 43\}+2=45$

$C_{max} = \gamma_4 = 45$

Nó 1243

$T=\{J_1, J_2, J_3, J_4\}$; $T'=\{\}$

α_3=24; β_3=max{24, 15}+12=36; γ_3=max{36, 22}+2=38

α_4=31; β_4=max{31, 36}+9=45; γ_4=max{45, 38}+13=58

$C_{max} = \gamma_4 = 58$

Nó 13XX

T={J_1, J_3}; T´={J_2, J_4}

α_2=10; β_2=max{10,7}+9=19; γ_2=max{19, 17}+13=32

lb(T)=max{10+21+6, 19+13+2, 32+7}=39

De igual forma, podemos obter o *Lower Bound* para os seguintes nós:

Nó 1324	**Nó 1342**	**Nó 14XX**
$C_{max} = 45$	$C_{max} = 39$	$C_{max} = 45$
Nó 2XXX	**Nó 3XXX**	**Nó 4XXX**
$C_{max} = 45$	$C_{max} = 46$	$C_{max} = 52$

Com os resultados obtidos anteriormente podemos completar a árvore, obtendo dessa forma a árvore apresentada na figura 5.9.

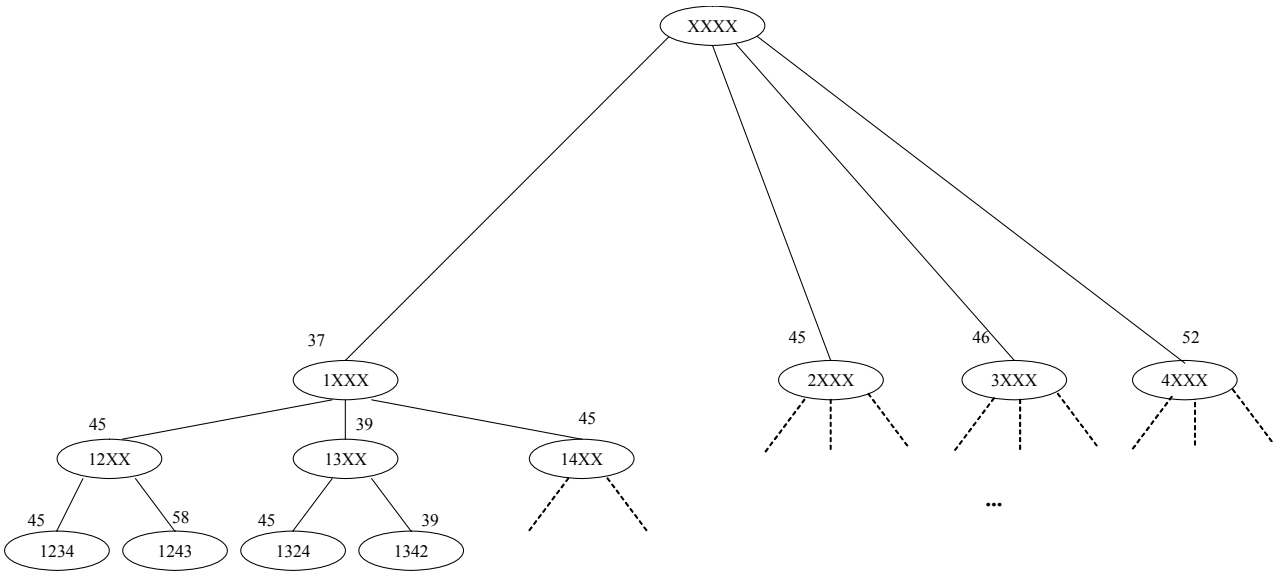


Figura 5.9 Árvore que representa os nós explorados e inexplorados para o problema 4/3/*F*/C_{max}.

Da figura 5.9 podemos concluir que a S_{opt} = (J_1, J_3, J_4, J_2) e C_{max} = 39.

5.9 Estudo computacional

Nesta secção apresentamos o software de sequenciamento desenvolvido para a Célula Flexível de Fabrico (CFF) [Ferrolho e Crisóstomo, 2005a], [Ferrolho e Crisóstomo, 2005b], [Ferrolho e Crisóstomo, 2005c] e [Ferrolho *et al.*, 2005]. Foi utilizada a linguagem de programação *C++ Builder* no desenvolvimento do referido software e na implementação dos algoritmos de sequenciamento [Alves, 2002] e [Kruglinski, 1997]. É feito um estudo computacional dos problemas de sequenciamento de máquina simples, duas máquinas e três máquinas, apresentados nas secções anteriores. O estudo destes problemas de sequenciamento tem diversas razões, nomeadamente de natureza metodológica, já que uma boa compreensão destes problemas ajudar-nos-á a compreender melhor os problemas de sequenciamento mais complexos. Uma vez que a CFF desenvolvida possui apenas duas máquinas, só foi possível transferir e testar nesta, problemas de sequenciamento até duas máquinas.

5.9.1 Arquitectura do software desenvolvido

O software de sequenciamento desenvolvido para a CFF apresenta a arquitectura mostrada na figura 5.10. A sequência das operações necessárias para desencadear uma ordem de fabrico na CFF, que iremos descrever a seguir, encontra-se esquematizada na figura 5.11. Foi utilizado o programa "InterBase" no desenvolvimento da base de dados "JOBS_SCHEDULING.GDB", sendo esta constituída por várias tabelas. Cada tabela possui informação relativa aos vários *jobs*, como por exemplo, tempos de processamento, datas de entrega, código CNC, imagem, etc.

A figura 5.11 mostra a sequência das operações necessárias para desencadear uma ordem de fabrico na CFF. Numa primeira fase, as operações decorrem no "PC central – Gestor da CFF" permitindo:

- Visualizar, actualizar, inserir e eliminar *jobs* na base de dados, caso seja necessário, através da opção "Data Base", como mostra a figura 5.12(a).
- Seleccionar o tipo de sequenciamento a efectuar, através da opção "Scheduling", como por exemplo, "Single Machine", "Two Machines", entre outros. Após a escolha do tipo de sequenciamento, surge uma janela semelhante à da figura 5.12(b) com os *jobs* disponíveis na base de dados e informação relativa aos mesmos.
- Obter a sequência óptima e o diagrama de *Gantt* dos *jobs* seleccionados, através da opção "Gantt Diagram", como mostra a figura 5.12(c).
- Transferir os códigos CNC dos *jobs* a maquinar para as máquinas CNC, através do botão "SEND PROGRAMS" da figura 5.12(d).

– Por último, dar ordem à CFF para iniciar a maquinagem dos *jobs*, através do botão "START" da figura 5.12(d).

Numa segunda fase, as operações decorrem nos quatro sectores da CFF, sendo estes responsáveis pela execução dos *jobs*. A partir do momento em que o operador pressiona o botão "START", a CFF irá maquinar os *jobs* de acordo com o diagrama de *Gantt* obtido na fase de sequenciamento. Os quatro sectores da CFF (PC1 - sector de fabrico, PC2 - sector de assemblagem, PC3 - sector de transporte e PC4 - sector de armazenamento) irão trabalhar de forma autónoma e coordenada, com o objectivo de se maquinarem todos os *jobs*.

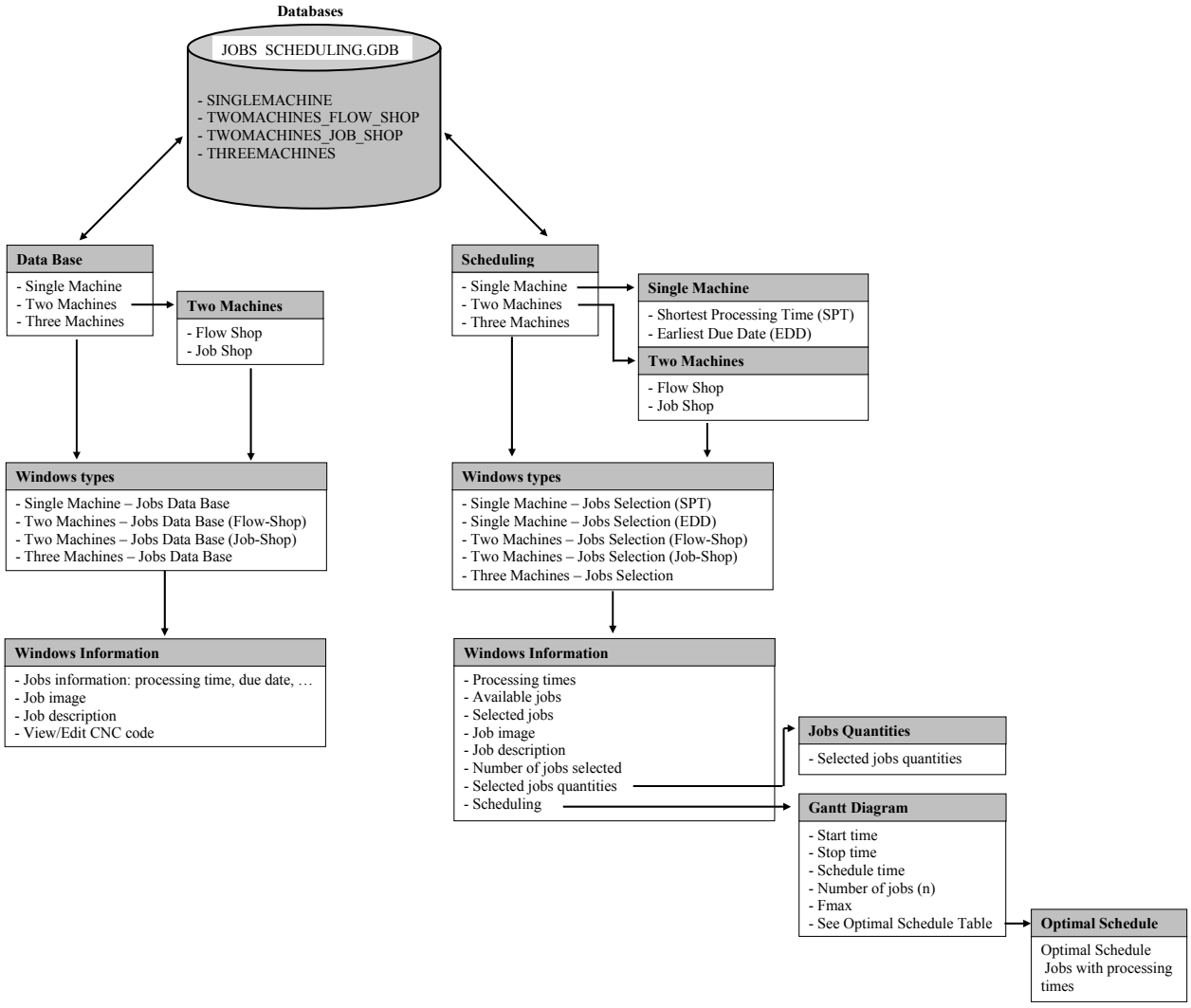


Figura 5.10 Arquitectura do software desenvolvido.

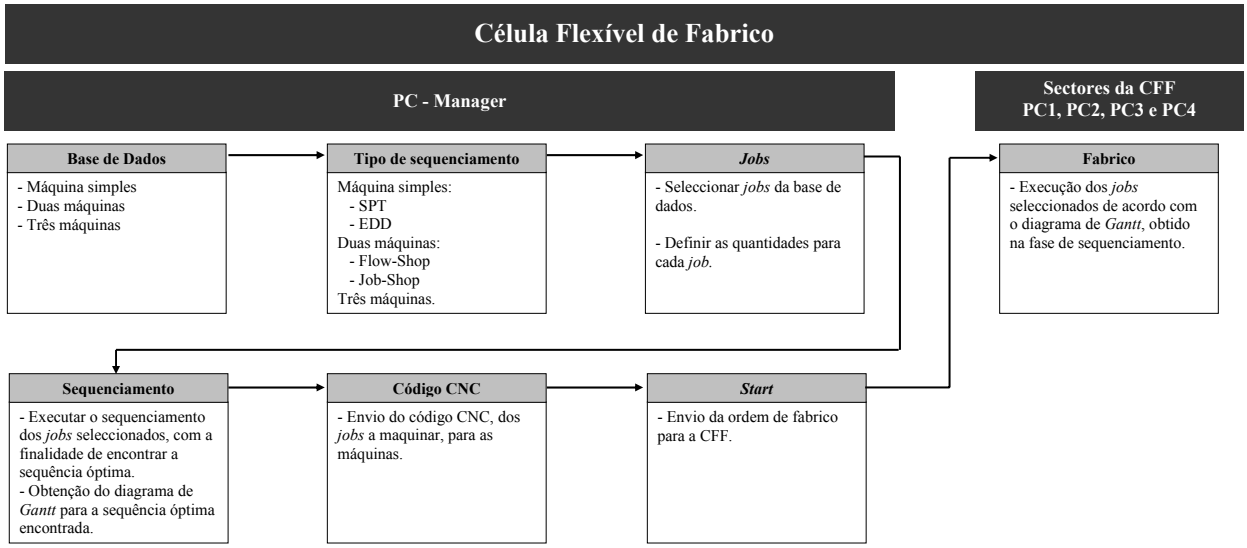
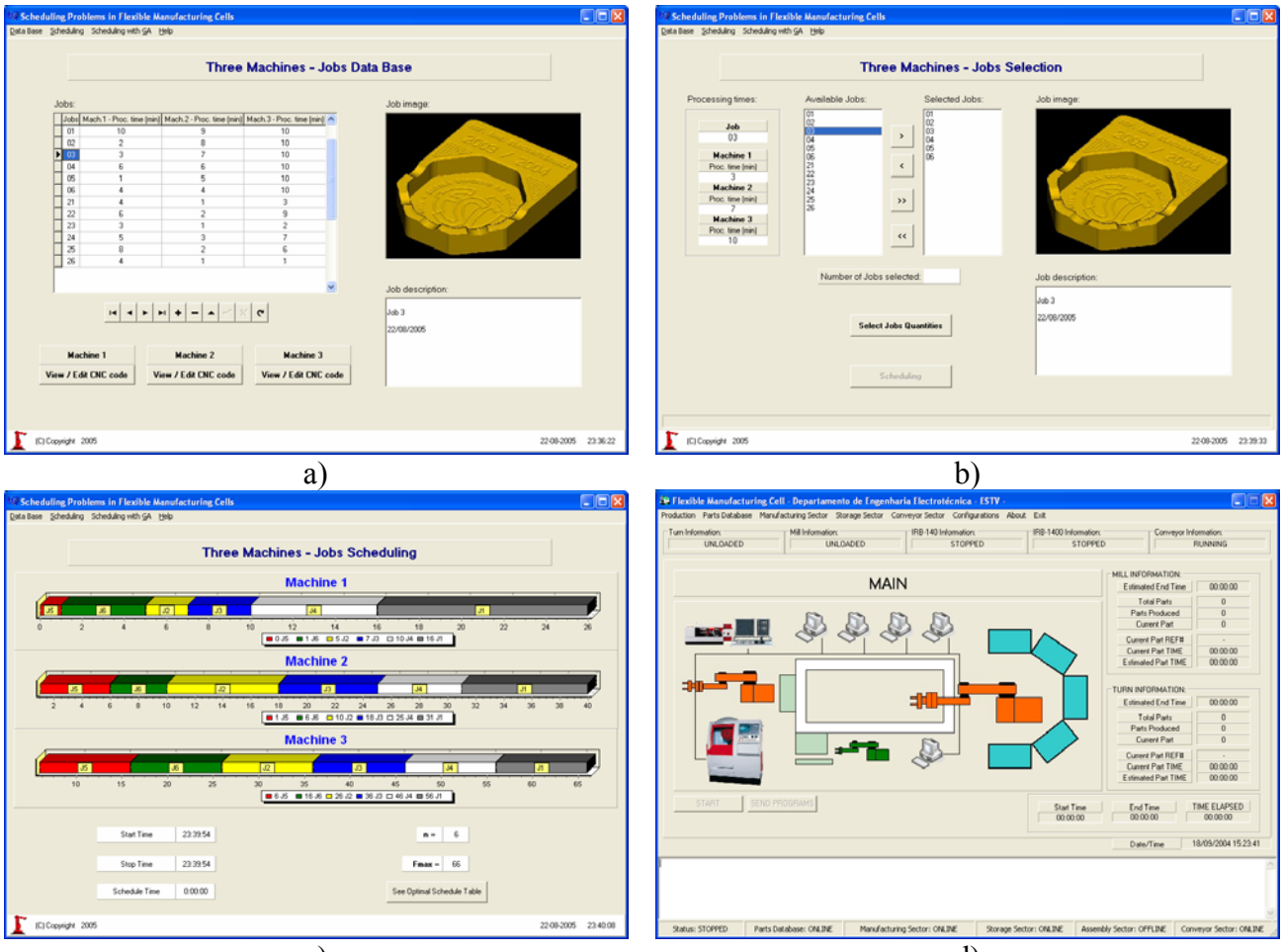


Figura 5.11 Sequência das operações na CFF.



a) b)

c) d)

Figura 5.12 Software desenvolvido:

a) Three Machines – Jobs Data Base
b) Three Machines – Jobs Selection
c) Three Machines – Jobs Scheduling
d) Janela principal da CFF – Flexible Manufacturing Cell.

5.9.2 Testes computacionais

Com a finalidade de testar e confirmar a eficiência dos algoritmos de sequenciamento, apresentados anteriormente, foi realizado um conjunto de testes. Estes, foram efectuados num computador com as seguintes características: *Pentium* IV, 1.7 GHz e 256 MB de RAM. Os testes computacionais apresentados nas tabelas 5.11 e 5.12 foram realizados durante os meses de Julho e Agosto de 2005. Foram utilizados os exemplos 5.1, 5.2, 5.3, 5.4 e 5.5 nos referidos testes computacionais. Durante esta fase de testes foram também usados outros exemplos, com outros *jobs*, para reforçar a eficiência dos algoritmos de sequenciamento e do software desenvolvido. No entanto, esses resultados não são apresentados nesta tese, por entendermos que os que se encontram nas tabelas 5.11 e 5.12 são suficientes para o estudo em questão.

Tabela 5.11 Resultados computacionais para os exemplos de máquina simples

Máquina simples			
SPT		**EDD**	
Jobs	Tempo do CPU hh:mm:ss	*Jobs*	Tempo do CPU hh:mm:ss
7	00:00:00	6	00:00:00
70	00:00:02	60	00:00:03
700	00:00:19	600	00:00:29
7.000	00:03:06	6.000	00:04:53
70.000	00:32:22	60.000	00:49:42
80.000	00:40:56	80.000	01:07:20
100.000	00:53:05	100.000	a)
120.000	00:59:57	120.000	
140.000	01:02:33	140.000	
160.000	a)	160.000	

a) Erro no computador – "*Out of memory*".

Os valores apresentados nas tabelas 5.11 e 5.12 mostram que o número máximo de *jobs* sequenciados foi:

- 140.000 *jobs*, para o exemplo de máquina simples e utilizando a regra SPT;
- 80.000 *jobs*, para o exemplo de máquina simples e utilizando a regra EDD;
- 80.000 *jobs*, para o exemplo de duas máquinas, sendo o problema do tipo *Flow Shop*;
- 140.000 *jobs*, para o exemplo de duas máquinas, sendo o problema do tipo *Job Shop*;
- 100.000 *jobs*, para o exemplo de três máquinas.

A partir destes valores foi obtido um erro no computador – "*Out of memory*". Não houve necessidade de ultrapassarmos o problema de memória ocorrido no computador, porque o

número máximo de *jobs* sequenciados em cada tipo de problema é, na nossa opinião, bastante significativo.

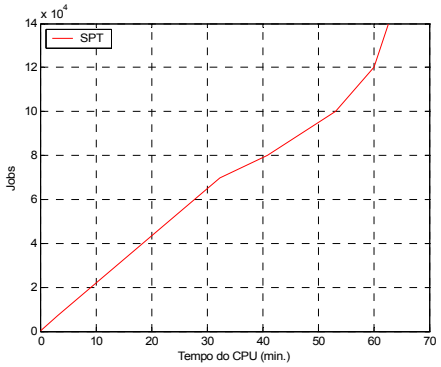
Tabela 5.12 Resultados computacionais para os exemplos de duas e três máquinas

Duas máquinas				**Três máquinas**	
Flow Shop		***Job Shop***		***Flow Shop***	
Jobs	Tempo do CPU hh:mm:ss	Jobs	Tempo do CPU hh:mm:ss	Jobs	Tempo do CPU hh:mm:ss
8	00:00:01	9	00:00:01	6	00:00:00
80	00:00:03	90	00:00:03	60	00:00:02
800	00:00:31	900	00:00:33	600	00:00:18
8.000	00:05:16	9.000	00:05:26	6.000	00:03:02
80.000	00:54:51	90.000	00:52:06	60.000	00:31:37
100.000	a)	100.000	00:58:55	100.000	00:59:14
120.000		120.000	01:10:53	120.000	a)
140.000		140.000	01:22:06	140.000	
160.000		160.000	a)	160.000	

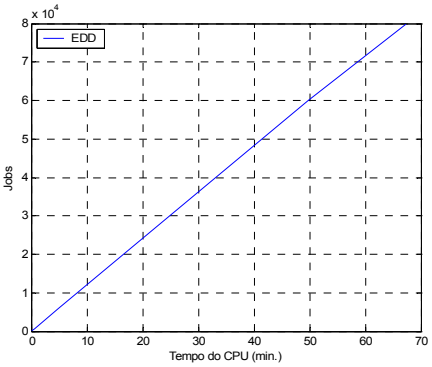
a) Erro no computador – "*Out of memory*".

Os gráficos apresentados na figura 5.13 mostram a relação entre o número de *jobs* a sequenciar e o tempo de processamento que o computador demorou a encontrar a sequência óptima. Estes gráficos foram obtidos utilizando os resultados computacionais apresentados nas tabelas 5.11 e 5.12. Dos gráficos apresentados na figura 5.13, podemos concluir que os algoritmos estudados e utilizados na resolução dos problemas de sequenciamento de máquina simples, duas máquinas e três máquinas são eficazes e eficientes. São eficazes porque nos permitem a obtenção de soluções de boa qualidade, como se demonstrou nos exemplos utilizados, onde se obtiveram soluções óptimas. São eficientes porque a complexidade temporal cresce polinomialmente e não exponencialmente com a dimensão do problema, permitindo assim a obtenção da solução em tempo útil. Fica, desta forma, demonstrado que estes algoritmos de sequenciamento pertencem à classe polinomial P, porque existe para cada um dos casos um algoritmo eficiente para a sua resolução. O gráfico da figura 5.13(f) permite efectuar uma comparação dos resultados computacionais obtidos para todas as máquinas (máquina simples, duas máquinas e três máquinas). Da análise do gráfico, podemos concluir que a regra SPT é a que processa um maior número de *jobs* e a regra EDD a que processa um menor número de *jobs* (para intervalos de tempo iguais).

Máquina simples

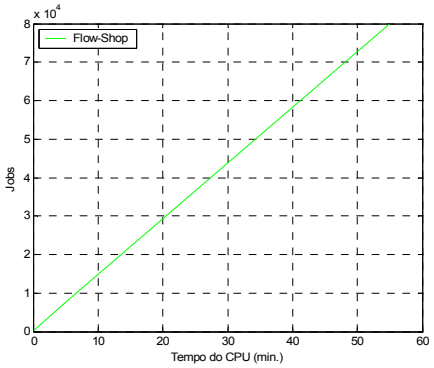


a)

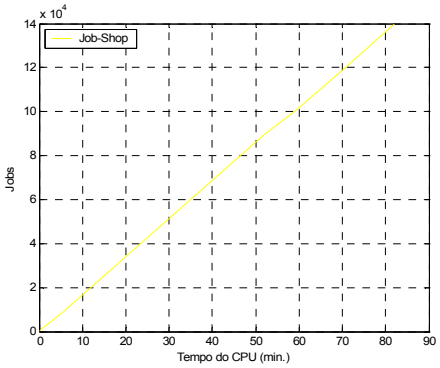


b)

Duas máquinas

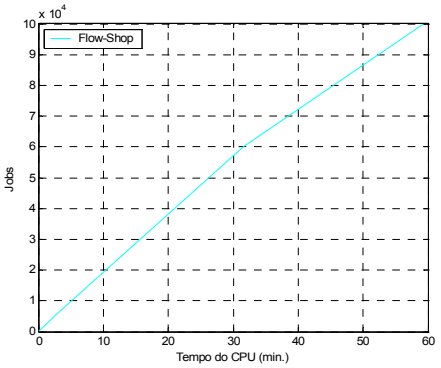


c)



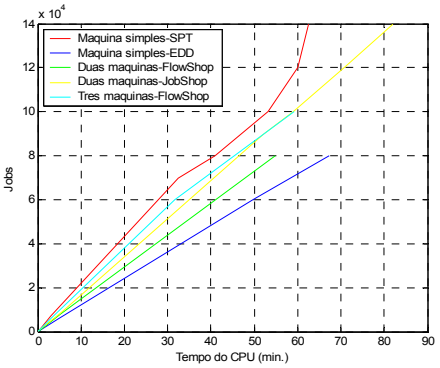
d)

Três máquinas



e)

Todas as máquinas



f)

Figura 5.13 Relação entre o número de *jobs* a sequenciar e o tempo do CPU:

a) Máquina simples - Regra SPT (*Shortest Processing Time Scheduling)*

b) Máquina simples - Regra EDD (*Earliest Due Date Scheduling*)

c) Duas máquinas - Algoritmo de *Johnson* para o problema *Flow-Shop*

d) Duas máquinas - Algoritmo de *Johnson* para o problema *Job-Shop*

e) Três máquinas - Algoritmo de *Johnson* para o problema *Flow-Shop*

f) Todas as máquinas

5.10 Resumo do Capítulo

Neste capítulo abordamos o problema de sequenciamento de tarefas em Células Flexíveis de Fabrico (CFF). Inicialmente, apresentamos o problema do sequenciamento, a terminologia e os conceitos utilizados ao longo desta tese. Foram abordados os problemas de sequenciamento em máquina simples, duas máquinas e três máquinas. Por último, foi feito o estudo computacional destes problemas de sequenciamento, onde apresentamos, também, a arquitectura do software desenvolvido e implementado na CFF, bem como os testes computacionais realizados. O estudo destes problemas de sequenciamento teve diversas razões, nomeadamente de natureza metodológica, já que uma boa compreensão destes problemas ajudar-nos-á a compreender melhor os problemas de sequenciamento mais complexos.

Os problemas de sequenciamento superiores a 3 *jobs*, *m* máquinas até *n jobs*, *m* máquinas, são de muito difícil resolução. A dificuldade está na computação pois existem $(n!)^m$ sequências possíveis, e cada uma tem que ser examinada para se seleccionar a que tiver o menor *makespan* ou aquela que for melhor segundo um outro critério de desempenho. Alguns autores desenvolveram alguns modelos de programação mas os resultados computacionais (complexidade computacional) não são encorajadores. Para pequenos problemas, a técnica *Branch and Bound* tem-se revelado muito boa, enquanto, para problemas de grande dimensão esta técnica é insustentável do ponto de vista computacional.

Capítulo 6

Algoritmos genéticos aplicados aos problemas de sequenciamento com uma só máquina

"A satisfação está no esforço feito para alcançar o objectivo, e não em tê-lo alcançado."
"Mahatma" Gandhi (1869-1948), líder político e espiritual Indiano.

6.1 Introdução

Os problemas reais de sequenciamento são de resolução muito complexa, sendo conhecidos em termos de teoria da complexidade como NP difíceis (*NP-hard*), tendo dificuldade não polinomial, para os quais o tempo requerido para a determinação de um plano óptimo ou sub-óptimo aumenta exponencialmente com o tamanho do problema [Morton *et al.*, 1993]. A dificuldade está na computação, pois existem $(n!)^m$ sequências possíveis, sendo n o número de *jobs* e m o número de máquinas. Cada sequência tem que ser examinada no sentido de se seleccionar a que tiver o menor *makespan* ou aquela que for melhor segundo um outro critério de desempenho. Alguns autores desenvolveram alguns modelos de programação mas os resultados computacionais (complexidade computacional) não são encorajadores. Para pequenos problemas, a técnica *Branch and Bound* tem-se revelado muito boa, enquanto para problemas de grande dimensão esta técnica é insustentável do ponto de vista computacional.

Em 1975, John Holland publicou o livro "Adaptation in Natural and Artificial Systems" onde apresenta e explica a teoria dos Algoritmos Genéticos (AG) [Holland, 1975]. No final da década de 80, David Goldberg aplicou os AG, pela primeira vez, à resolução de problemas industriais [Goldberg, 1989] e desde então estes têm sido aplicados aos mais diversos tipos de problemas. Os AG baseiam-se nos princípios Darwinianos da evolução natural das espécies e podem ser vistos como um método de resolução automática de problemas que obtém soluções através da combinação de outras soluções. Os recentes avanços têm permitido a utilização dos AG na resolução de problemas industriais, nomeadamente ao nível da resolução de problemas de sequenciamento (*scheduling*). Porém, um AG na sua forma básica é insuficiente para a resolução

deste tipo de problemas [Syswerda, 1991], [Murata *et al.*, 1994]. Por exemplo a codificação *bit--string* é inadequada; a selecção através da proporcionalidade do *fitness* conduz à convergência prematura; alguns operadores de cruzamento e mutação tornam as soluções descendentes inviáveis, e ainda, um ajuste inadequado dos parâmetros pode diminuir a performance global do AG. Por este motivo, as decisões a tomar na elaboração de um AG devem ser analisadas cuidadosamente, uma vez que existem muitas possibilidades de implementação.

Este capítulo encontra-se organizado da seguinte forma: a secção 6.2 apresenta um novo conceito de operadores genéticos para problemas de sequenciamento; a secção 6.3 apresenta o software desenvolvido HybFlexGA, usado para examinar a performance dos vários operadores de cruzamento e de mutação; a secção 6.4 aborda os problemas de sequenciamento com uma só máquina; a secção 6.5 mostra os resultados dos testes computacionais obtidos no exame e avaliação dos operadores genéticos; a secção 6.6 mostra os resultados computacionais obtidos em problemas reais de sequenciamento com 40, 50 e 100 *jobs*; a secção 6.7 descreve e apresenta a arquitectura desenvolvida para a resolução de problemas dinâmicos de sequenciamento, baseados em algoritmos genéticos e a secção 6.8 apresenta as conclusões deste capítulo.

6.2 Operadores Genéticos

Quando são utilizados AG na resolução de problemas de sequenciamento, podem ser usados vários operadores de cruzamento e mutação. Assim, para desenvolvermos AG de alta performance é necessário escolher cuidadosamente estes operadores genéticos, bem como as respectivas probabilidades de cruzamento e de mutação.

Nesta secção iremos apresentar um conjunto de novos operadores genéticos, desenvolvidos no âmbito desta tese, com o objectivo de serem utilizados na resolução de problemas de sequenciamento [Ferrolho e Crisóstomo, 2006a].

6.2.1 Operadores de Cruzamento

Nesta subsecção apresentaremos os operadores de cruzamento desenvolvidos no âmbito desta tese de doutoramento. Sucintamente, o cruzamento é uma operação que gera uma nova sequência (cromossoma Filho) a partir de duas sequências (cromossomas Pai 1 e Pai 2). Assim, os operadores de cruzamento desenvolvidos são [Ferrolho e Crisóstomo, 2006a]:

- *one-point crossover: 1 child*;

- *two-point crossover: 1 child (version I)*;
- *two-point crossover: 1 child (version II)*;
- *one-point crossover: 2 children*;
- *two-point crossover: 2 children (version I)*;
- *two-point crossover: 2 children (version II)*;
- *two-point crossover: 3 children (version I)*;
- *two-point crossover: 3 children (version II)*;
- *two-point crossover: 4 children*.

Com o objectivo de compararmos a performance dos operadores de cruzamento por nós desenvolvidos, foram seleccionados, da literatura, os melhores operadores de cruzamento em problemas de sequenciamento. Assim, seleccionamos os seguintes operadores de cruzamento:

- *order crossover* [Goldberg, 1989];
- *cycle crossover* [Oliver *et al.*, 1987];
- *position based crossover* [Syswerda, 1991].

Em [Goldberg, 1989] e [Oliver *et al.*, 1987] são apresentados resultados teóricos e empíricos que comparam os operadores *partially matched crossover* (PMX), *order crossover* (OX) e *cycle crossover* (CX). Eles concluem que o PMX e o OX são semelhantes, embora o PMX tenda a preservar a posição absoluta do gene (devido ao mapeamento ponto a ponto) e o OX tenda a preservar a posição relativa do gene (devido ao preenchimento sequencial dos espaços vazios). Tendo em consideração as conclusões a que chegaram Goldberg e Oliver, optámos pelo OX. Também optámos por seleccionar o operador de cruzamento *position based crossover* (PBX), uma vez que vários autores conseguiram bons resultados aquando da sua aplicação a problemas de sequenciamento.

One-point crossover: 1 child

O operador de cruzamento *one-point crossover: 1 child* (OPC1C) está ilustrado na figura 6.1. É seleccionado aleatoriamente um ponto de corte, com a finalidade de dividir o cromossoma Pai 1 em duas partes. Os *jobs* localizados à esquerda do ponto do corte (o lado esquerdo e o lado direito são escolhidos de forma aleatória) são herdados pelo cromossoma Filho e os restantes *jobs* serão herdados do Pai 2, e colocados no cromossoma Filho segundo a mesma ordem. O cromossoma Pai, onde é efectuado o ponto de corte, é seleccionado aleatoriamente.

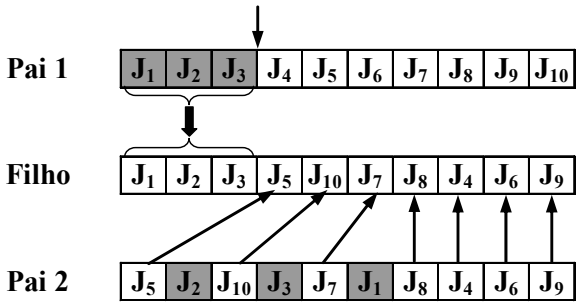


Figura 6.1 *One-point crossover: 1 child*.

Two-point crossover: 1 child (Version I)

O operador de cruzamento *two-point crossover: 1 child (Version I)* (TPC1CV1) está ilustrado na figura 6.2. Neste operador de cruzamento, são seleccionados aleatoriamente dois pontos de corte com o objectivo de dividir um dos cromossomas progenitores (Pai 1 ou Pai 2). Os *jobs* que se encontram fora dos pontos de corte serão herdados pelo cromossoma descendente (Filho). Os restantes genes em falta no cromossoma Filho serão herdados do cromossoma Pai 2, e colocados segundo a mesma ordem.

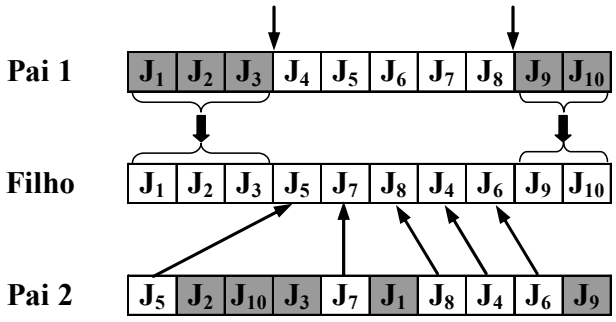


Figura 6.2 *Two-point crossover: 1 child (Version I)*.

Two-point crossover: 1 child (Version II)

O operador de cruzamento *two-point crossover: 1 child (Version II)* (TPC1CV2) está ilustrado na figura 6.3. Este operador de cruzamento é muito semelhante ao descrito anteriormente. No TPC1CV2 os *jobs* que se encontram dentro dos pontos de corte serão herdados pelo cromossoma descendente (Filho). Os restantes *jobs* em falta serão herdados do cromossoma Pai 2.

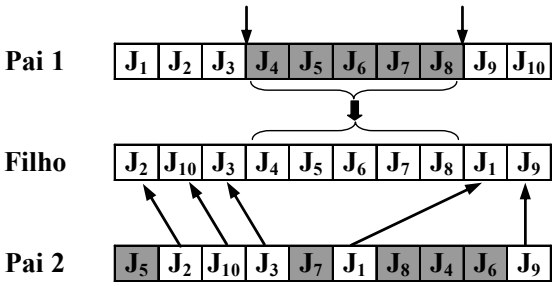


Figura 6.3 *Two-point crossover: 1 child (Version II)*.

One-point crossover: 2 children

A figura 6.4 mostra o operador *one-point crossover: 2 children* (OPC2C). Como se pode observar, este operador de cruzamento gera dois descendentes (Filho 1 e Filho 2). Os descendentes são gerados da mesma forma que no operador OPC1C, descrito anteriormente.

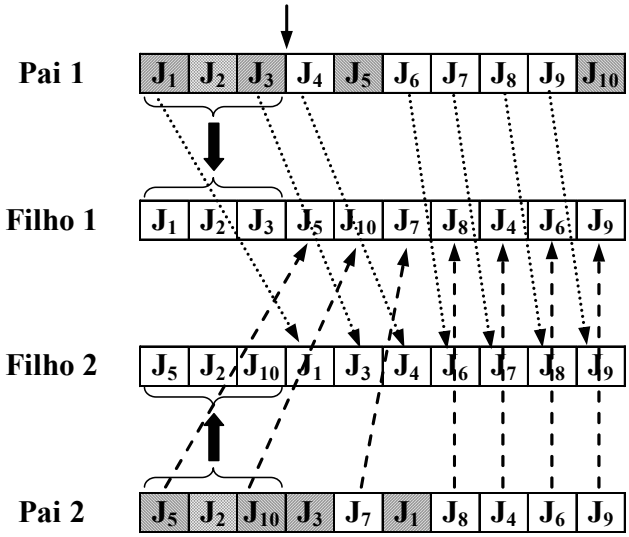


Figura 6.4 *One-point crossover: 2 children*.

Two-point crossover: 2 children (Version I)

A figura 6.5 mostra o operador *two-point crossover: 2 children (Version I)* (TPC2CV1). Este operador de cruzamento, como o próprio nome indica, também gera dois descendentes (Filho 1 e Filho 2). Os descendentes são gerados da mesma forma que no operador TPC1CV1, descrito anteriormente.

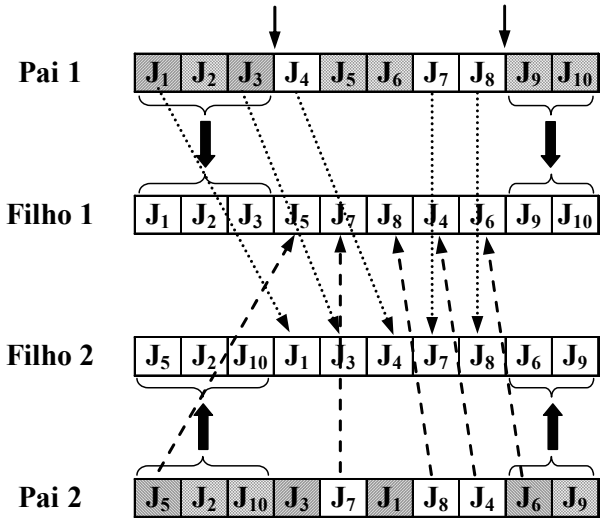


Figura 6.5 *Two-point crossover: 2 children (Version I)*.

Two-point crossover: 2 children (Version II)

A figura 6.6 mostra o operador *two-point crossover: 2 children (Version II)* (TPC2CV2). Este operador de cruzamento também gera dois descendentes (Filho 1 e Filho 2). Os descendentes são gerados da mesma forma que no operador TPC1CV2, descrito anteriormente.

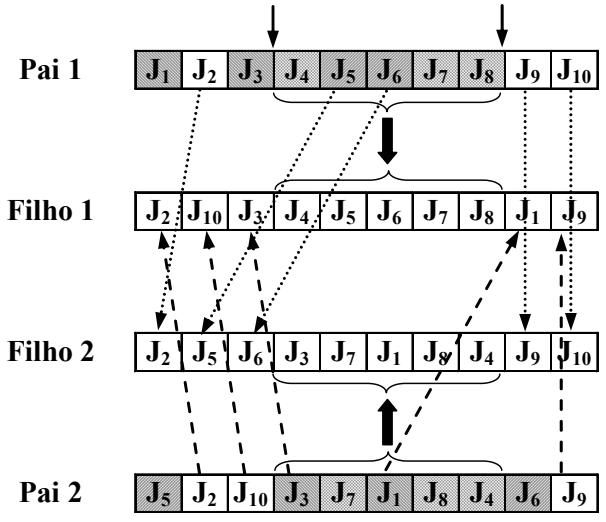


Figura 6.6 *Two-point crossover: 2 children (Version II)*.

Two-point crossover: 3 children (Version I)

O operador de cruzamento *two-point crossover: 3 children (Version I)* (TPC3CV1) é uma mistura dos operadores de cruzamento TPC1CV1 e TPC2CV1. Sempre que se aplica este operador de cruzamento são gerados 3 cromossomas filhos.

Two-point crossover: 3 children (Version II)

O operador de cruzamento *two-point crossover: 3 children (Version II)* (TPC3CV2) é uma mistura dos operadores de cruzamento TPC1CV2 e TPC2CV2. Sempre que se aplica este operador de cruzamento são gerados 3 cromossomas filhos.

Two-point crossover: 4 children

O operador de cruzamento *two-point crossover: 4 children* (TPC4C) é uma mistura dos operadores de cruzamento TPC2CV1 e TPC2CV2. Sempre que se aplica este operador de cruzamento são gerados 4 cromossomas filhos.

Order crossover

Consideremos os dois progenitores apresentados na figura 6.7, onde se encontram dois pontos de corte assinalados por duas setas.

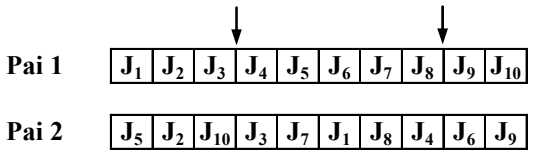


Figura 6.7 Progenitores: Pai 1 e Pai 2.

Considerando L como a localização de um gene livre e admitindo que a parte central se mantém, teremos para o exemplo citado as sequências apresentadas na figura 6.8.

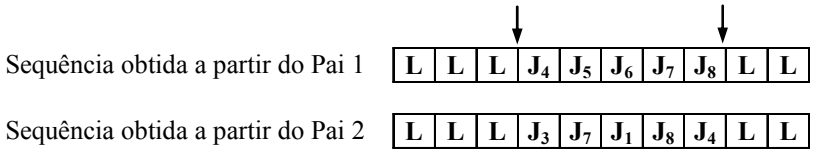


Figura 6.8 Sequências com a localização dos genes livres.

Partindo do segundo ponto de corte iremos ver as sequências que ocorrem no outro progenitor a partir desse ponto:

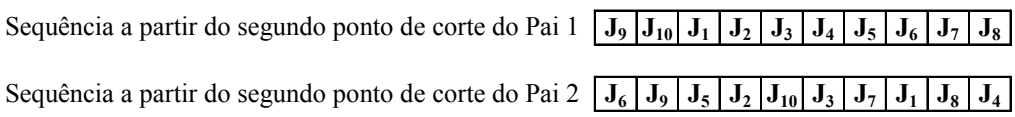


Figura 6.9 Sequências a partir do segundo ponto de corte.

Vamos agora eliminar os *jobs* conhecidos na parte central do cromossoma Pai 1 (J_4, J_5, J_6, J_7 e J_8) na sequência a partir do 2º ponto de corte do Pai 2 e os j*obs* conhecidos na parte central do cromossoma do Pai 2 (J_3, J_7, J_1, J_8 e J_4) na sequência a partir do 2º ponto de corte do Pai 1, originando as sequências das figuras 6.10 e 6.11.

Figura 6.10 Sequência resultante a partir do 2º ponto de corte do Pai 2.

Figura 6.11 Sequência resultante a partir do 2º ponto de corte do Pai 1.

Tais sequências vão preencher as posições livres (L) anteriormente referidas, a partir do segundo ponto de corte, originando os descendentes Filho 1 e Filho 2 apresentados na figura 6.12.

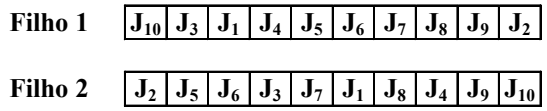


Figura 6.12 Descendentes: Filho 1 e Filho 2.

Através da figura 6.12, verificamos que os j*obs* entre os dois pontos de corte mantiveram a sua ordem relativamente aos cromossomas progenitores (Pai 1 e Pai 2). Por isso, o operador de cruzamento o*rder crossover* é muitas vezes referido na literatura portuguesa como "operador de cruzamento com manutenção da ordem".

Cycle crossover

O operador *cycle crossover* (CX) não utiliza pontos de corte, mas gera descendentes (filhos) com elevado valor genético [Oliver *et al.*, 1987] e [Goldberg, 1989]. Consideremos a figura 6.13 a), onde a posição de início no cromossoma Pai 1 é aleatoriamente seleccionada. O gene J_1 do Pai 1 é copiado para o cromossoma Filho. Como mostra a figura 6.13 b), o Filho vai herdar o gene J_9 do Pai 1, porque o J_9 ocupa a primeira posição no Pai 2. Por sua vez, o gene J_9 ocupa a posição 9 no cromossoma do Pai 1, e a mesma posição do cromossoma do Pai 2 é ocupada pelo gene J_2. Assim, o cromossoma Filho herda para a posição 2 o gene J_2 do Pai 1, como mostra a figura 6.13 c). Este ciclo continua até à figura 6.13 f), onde o cromossoma Filho herda o gene J_8 que se

encontra na posição 8 do Pai 1. Como o gene que ocupa a posição 8 do Pai 2 (J_1) já se encontra no Filho, este passa a herdar os genes do Pai 2. Na figura 6.13 g) o Filho herda o gene que se encontra na posição 3 do Pai 2 (J_5). O ciclo continua até estar completo o cromossoma Filho.

Da mesma forma, poderia ser obtido um segundo descendente (Filho 2) e assim passaríamos a ter dois descendentes sempre que se aplica o operador CX.

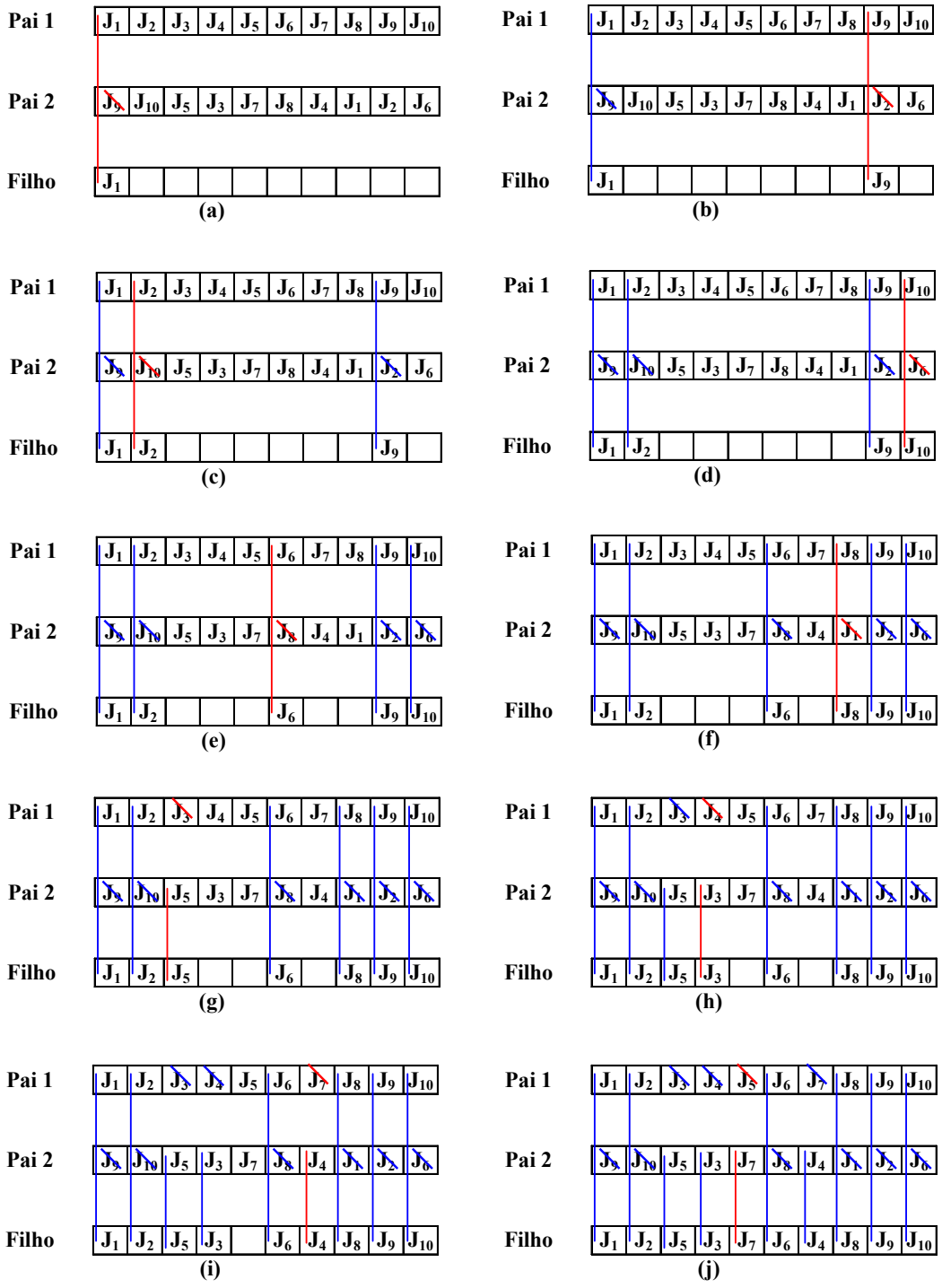


Figura 6.13 *Cycle crossover*.

Position based crossover

O operador de cruzamento *position based crossover* (PBX) está ilustrado na figura 6.14 [Syswerda, 1991]. À semelhança do operador de cruzamento CX, este operador não possui pontos de corte. Os genes seleccionados aleatoriamente e marcados com um "*" são herdados pelo cromossoma Filho. O número de "*" deve ser um número inteiro aleatório e pertencer ao intervalo [1, *n*], sendo *n* o número de *jobs*.

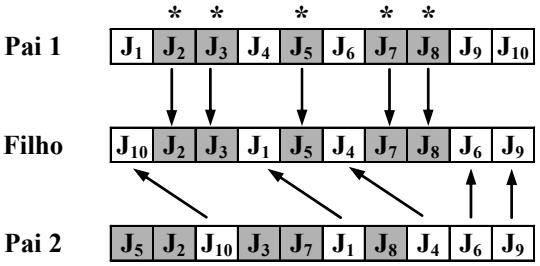


Figura 6.14 *Position based crossover*.

6.2.2 Operadores de Mutação

O objectivo da mutação é assegurar a diversidade genética de uma população de cromossomas, consistindo na mudança aleatória de um ou mais genes num cromossoma descendente. Os operadores de mutação mais conhecidos para problemas de sequenciamento referem-se ao *swap*, que consiste na troca de posição de dois genes aleatoriamente seleccionados, e ao *shift*, que se baseia no deslocamento de um gene para a direita ou para esquerda em algumas posições. Em [Murata *et al.*, 1996] é apresentado um estudo computacional do desempenho de cinco operadores de mutação na resolução de problemas de sequenciamento, tendo sido obtidos melhores resultados com operador *shift*.

Existem vários operadores de mutação largamente divulgados na literatura, sendo os mais importantes [Murata *et al.*, 1996]:

- *adjacent two-job change* (Adj2JC);
- *arbitrary two-job change* (Arb2JC);
- *arbitrary three-job change* (Arb3JC);
- *shift change* (SC).

Adjacent two-job change

A figura 6.15 apresenta o operador de mutação *adjacent two-job change* (Adj2JC). Este operador selecciona aleatoriamente dois *jobs* adjacentes e de seguida efectua a troca de posição dos mesmos.

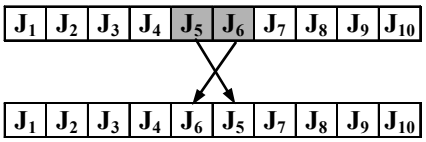


Figura 6.15 *Adjacent two-job change*.

Arbitrary two-job change

A figura 6.16 apresenta o operador de mutação *arbitrary two-job change* (Arb2JC). Este operador de mutação selecciona aleatoriamente dois *jobs* e de seguida efectua uma troca de posição dos mesmos. O operador de mutação *adjacent two-job change* é um caso particular deste operador de mutação.

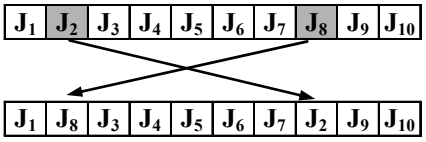


Figura 6.16 *Arbitrary two-job change*.

Arbitrary three-job change

A figura 6.17 apresenta o operador de mutação a*rbitrary three-job change* (Arb3JC). Como mostra a figura 6.17, são seleccionados aleatoriamente três *jobs* para de seguida serem trocadas as posições desses *jobs*.

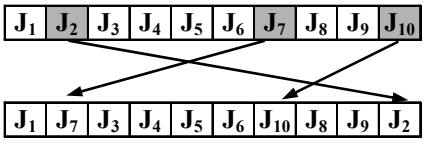


Figura 6.17 *Arbitrary three-job change*.

Shift change

O operador de mutação *shift change* (SC) encontra-se esquematizado na figura 6.18. Este operador escolhe aleatoriamente um *job* e de seguida coloca-o numa nova posição, também esta

escolhida aleatoriamente. Por último, os *jobs* situados entre as duas posições irão sofrer um deslocamento.

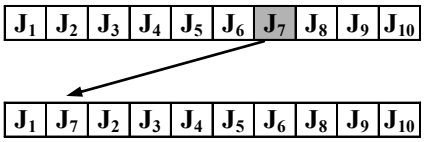


Figura 6.18 *Shift change*.

Arbitrary 20%-job change

Tendo verificado que os operadores de mutação descritos atrás são rígidos, isto é, não têm em conta o tamanho do cromossoma, desenvolvemos operadores de mutação em que o número de *jobs* que sofrem mutação é proporcional ao tamanho do cromossoma. Depois de se terem efectuado vários testes computacionais, variando a percentagem de *jobs* para mutação, escolheu-se o operador de mutação *arbitrary 20%-job change* (Arb20%JC). A figura 6.19 apresenta o operador de mutação Arb20%JC, desenvolvido no âmbito desta tese. Neste operador de mutação, 20% dos *jobs* são seleccionados aleatoriamente para de seguida mudarem de posições. A grande vantagem deste operador de mutação é a sua flexibilidade, isto é, o número de *jobs* que irá sofrer mutação depende do tamanho do cromossoma. Por exemplo, se estivermos perante um cromossoma de 40 *jobs* e um outro de 100 *jobs*, o número de *jobs* que irá sofrer mutação é de 8 e 20, respectivamente. Esta flexibilidade é muito importante nos problemas de sequenciamento, porque o número de *jobs* a sequenciar varia com muita frequência. Assim, a percentagem usada por este operador permite que se adapte ao número de *jobs* que irá ser sequenciado [Ferrolho e Crisóstomo, 2006a], [Ferrolho e Crisóstomo, 2006e].

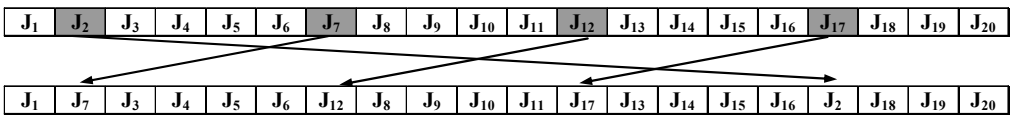


Figura 6.19 *Arbitrary 20%-job change*.

6.3 Software desenvolvido

Com o objectivo de examinarmos a performance dos vários operadores de cruzamento e de mutação apresentados na secção anterior, desenvolvemos o software HybFlexGA (*Hybrid and Flexible Genetic Algorithm*). A figura 6.20 apresenta a arquitectura do HybFlexGA, estando esta dividida em 3 módulos: o módulo de interface gráfica, o módulo de pré-processamento e o módulo de sequenciamento com AG [Ferrolho e Crisóstomo, 2006c]. Esta aplicação, à

semelhança das outras desenvolvidas nesta tese, foi desenvolvida utilizando a linguagem de programação C++ Builder [Reisdorph e Henderson, 1997] e [Calvert, 1997].

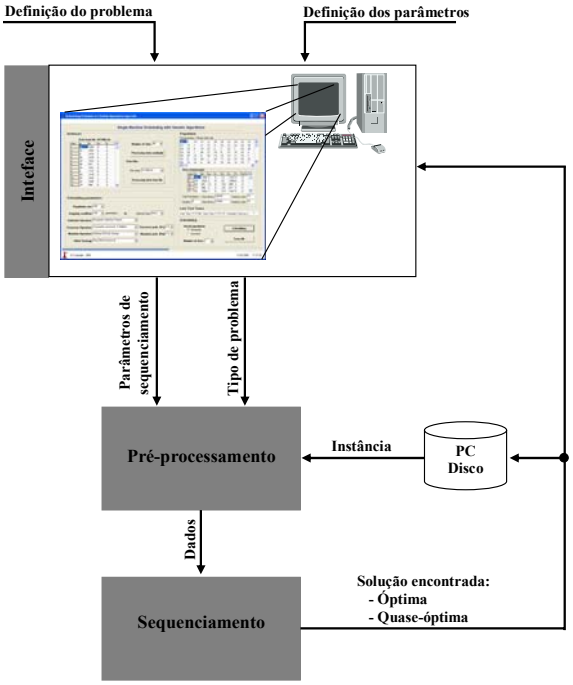


Figura 6.20 Arquitectura do software proposto.

Módulo de interface com o utilizador

O módulo de interface com o utilizador é determinante para o sucesso dum sistema de sequenciamento. Assim, esta interface deve ser amigável e dinâmica de modo a permitir aceder e manipular facilmente a informação relativa ao plano de sequenciamento, aos *jobs*, aos operadores genéticos, entre outras. Esta interface é do tipo *Graphical User Interface* (GUI) e permite a interactividade entre o utilizador e o módulo de sequenciamento com AG, permitindo uma fácil introdução de dados e a visualização das soluções encontradas. A figura 6.21 mostra a interface desenvolvida.

Módulo de pré-processamento

O módulo de pré-processamento trata a informação vinda do módulo da interface com o utilizador, que inclui o tipo de problema e os parâmetros de sequenciamento necessários ao módulo seguinte – módulo de sequenciamento com AG. Assim, o módulo de pré-processamento trata a informação de entrada do sistema, como a definição da instância do problema e a parametrização do AG utilizado no módulo seguinte. A instância do problema de sequenciamento pode ser gerada aleatoriamente ou a partir dum ficheiro do disco do PC, como mostra a figura 6.20.

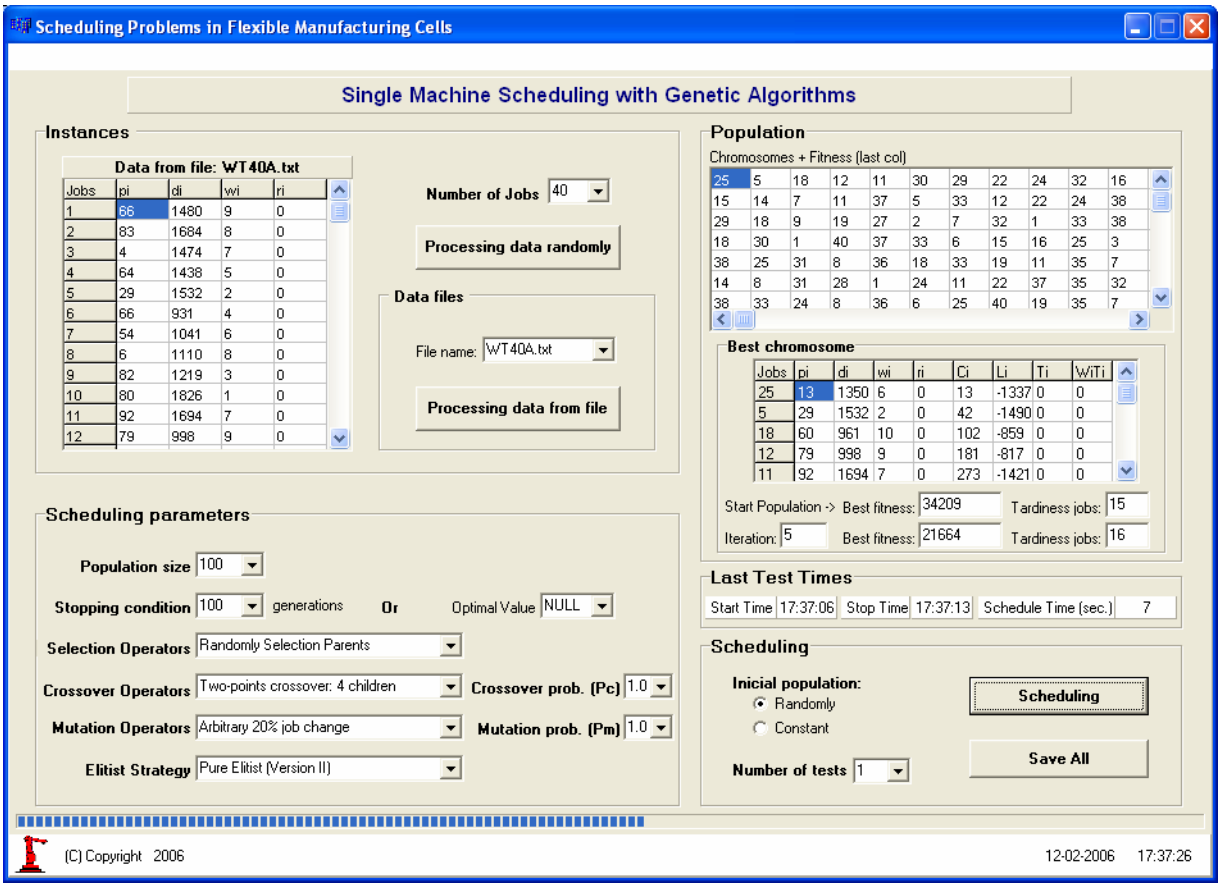


Figura 6.21 Interface desenvolvida.

Módulo de sequenciamento com AG

O módulo de sequenciamento com AG tem como objectivo a aplicação do algoritmo genético para encontrar a solução óptima do problema de sequenciamento. Quando a solução óptima não é encontrada, o algoritmo devolve a melhor solução obtida (solução satisfatória, também designada, muitas vezes, de quase-óptima). A figura 6.22 mostra o algoritmo genético proposto e implementado neste módulo.

1º Passo (Inicialização)

Colocar t=0, sendo t o índice da geração, e gerar aleatoriamente uma população inicial Ψ_0, incluindo N_{pop} soluções (N_{pop} é o número de soluções em cada população, i. e., N_{pop} é o tamanho da população). De referir que o número de soluções (cromossomas) N_{pop} na geração t é dada por $\Psi_t = \left\{x_t^1, x_t^2, ..., x_t^{N_{pop}}\right\}$.

2º Passo (Selecção)

Seleccionar pares de soluções (cromossomas pais) da população corrente Ψ_t de acordo com a probabilidade de cruzamento (P_c). Cada cromossoma x_t^i é seleccionado de acordo com o operador de selecção escolhido no software.

3º Passo (Cruzamento)

Aplicar o operador de cruzamento seleccionado no software aos pares de cromossomas gerados no 2º passo. Desta forma, serão gerados novos cromossomas de acordo com a probabilidade de cruzamento (P_c) escolhida.

4º Passo (Mutação)

Aplicar o operador de mutação seleccionado no software aos cromossomas gerados no 3º passo, de acordo com a probabilidade de mutação (P_m) escolhida.

5º Passo (Elitismo)

São seleccionados N_{pop} melhores cromossomas para formarem a população seguinte Ψ_{t+1} e os restantes cromossomas são eliminados. Assim, alguns cromossomas pais têm a possibilidade de passar para a geração seguinte. De seguida, verifica-se se existem cromossomas repetidos em Ψ_{t+1} e caso existam são substituídos por novos cromossomas.

6º Passo (Teste de finalização)

Se a condição de paragem seleccionada no software se verificar então pára a execução do algoritmo. Caso contrário, actualiza t para $t:=t+1$ e volta ao 2º passo. Nos testes computacionais realizados a condição de paragem utilizada foi o número de gerações.

Figura 6.22 Algoritmo genético implementado.

Aquando do desenvolvimento do software HybFlexGA tivemos a preocupação de desenvolver uma ferramenta de trabalho flexível, robusta e do tipo *user friendly*. Mas, a preocupação principal, tendo em conta os testes computacionais que pretendíamos realizar, foi a de desenvolver uma ferramenta de trabalho capaz de dar resposta às situações mais exigentes e variadas. Assim, o software desenvolvido permite:

- Seleccionar o número de *jobs* a sequenciar e gerar aleatoriamente os dados necessários ao sequenciamento (p_i, d_i, w_i, r_i). Também é possível gerar instâncias de problemas a partir de ficheiros.
- Seleccionar os parâmetros a serem utilizados pelo algoritmo genético, como por exemplo, o tamanho da população, a condição de paragem, operador de selecção, operador de cruzamento e a probabilidade de cruzamento (P_c), operador de mutação e a probabilidade de mutação (P_m) e elitismo.
- Visualizar os cromossomas utilizados na população inicial, assim como a população final obtida. É possível visualizar o *fitness* e o número de *jobs* atrasados na população inicial e final, bem como visualizar o número da geração onde o melhor cromossoma foi encontrado.

- Visualizar os tempos de sequenciamento, como por exemplo, o início, o fim e o tempo total do sequenciamento.
- Seleccionar o número de testes que se pretendem efectuar e a forma como a população inicial deve ser gerada (aleatoriamente ou constante).
- Guardar em ficheiros de texto (*.txt) informação relevante sobre o comportamento do algoritmo genético, como por exemplo, a média do *fitness*, o desvio padrão, o valor do *fitness* do melhor e do pior cromossoma em cada geração e a média do tempo do CPU.

Inicialmente, o software por nós desenvolvido irá ser utilizado na avaliação da performance e desempenho dos operadores genéticos apresentados na secção 6.2. Pretendemos com este estudo verificar quais são os operadores genéticos mais eficientes, na resolução de problemas de sequenciamento. Serão também feitos testes computacionais no sentido de averiguarmos a influência dos parâmetros de sequenciamento na qualidade da solução final. Por exemplo, o tamanho da população, a probabilidade de cruzamento, a probabilidade de mutação, são alguns dos parâmetros a ter em conta nesses testes computacionais.

6.4 Problemas de sequenciamento com uma só máquina

Muitos dos problemas de sequenciamento com uma só máquina pertencem à classe dos problemas de difícil resolução (NP-dificeis), isto é, com dificuldade não polinomial, para os quais não é ainda conhecido nenhum algoritmo determinístico eficiente que os resolva em tempo útil [Brucker, 2001].

Por vezes, num problema com várias máquinas, são resolvidos de forma independente problemas mais elementares, e só depois é incorporado o resultado no problema principal. Por exemplo, em modelos com várias máquinas, pode existir uma máquina cuja capacidade de processamento seja inferior à necessária, designada habitualmente por máquina crítica ou gargalo. A sua análise e tratamento como problema de uma só máquina pode ser útil nas futuras decisões de sequenciamento [Goldratt e Fox, 1986] e [Adams *et al.*, 1988]. Esta ideia pode ser estendida para os sistemas de fabrico constituídos por recursos geograficamente distribuídos ou até às empresas virtuais [Camarinha-Matos e Afsarmanesh, 1999].

O problema de sequenciamento de *n jobs* numa só máquina é caracterizado pelas seguintes restrições ou pressupostos:

- Os *jobs* (i=1, 2, ..., n) são independentes e caracterizados por tempos de processamento p_i. O caso particular de todos os tempos de processamento serem unitários (p_i=1 para todos os *jobs*) conduz, geralmente, a problemas de mais fácil resolução.
- No instante de tempo zero (t=0) a máquina está disponível para o processamento de um conjunto de n *jobs* independentes.
- Os tempos de preparação dos *jobs* são independentes da sequência escolhida e podem ser considerados no tempo de processamento.
- Os tempos de processamento e as datas de entrega dos *jobs* são conhecidos previamente.
- A máquina está sempre disponível.
- Uma vez iniciado o processamento de um *job*, este é processado até ao fim sem interrupções.

O número total de soluções diferentes para o problema de uma só máquina é $n!$. As técnicas definidas e testadas para este tipo de problemas constituem frequentemente ferramentas eficientes para a resolução de problemas mais gerais, de diferentes classes de problemas de sequenciamento.

6.4.1 Minimização da Soma dos Atrasos Pesados

O problema de minimização da soma dos atrasos pesados com uma só máquina (*Single Machine Total Weighted Tardiness Problem* – SMTWT) consiste em sequenciar n *jobs* numa só máquina, sem interrupção. A cada *job i* está associado um tempo de processamento p_i, uma data de entrega d_i e uma penalização w_i por cada unidade de tempo em atraso. O objectivo é encontrar uma sequência de *jobs* que minimize os atrasos pesados por w_i. O processamento da primeira tarefa da sequência começa no instante de tempo t=0. O atraso de um *job* na sequência é dado por $T_i=max\{0,\ C_i-d_i\}$, sendo C_i o instante de finalização do *job i*. Assim, pretende-se encontrar a sequência de processamento dos *jobs* que conduz à minimização de $\sum w_i T_i$, isto é, $min\sum w_i T_i$.

O problema de sequenciamento com uma só máquina é de difícil resolução, sendo conhecido em termos de teoria da complexidade como pertencente à classe de problemas NP difícil (*NP-hard*), com dificuldade não polinomial, onde o tempo requerido para a determinação de um plano óptimo aumenta exponencialmente com a dimensão do problema [Lawler, 1977], [Morton *et al.*, 1993] e [Brucker, 2001]. Na literatura têm sido propostas várias abordagens para a resolução deste tipo de problemas, baseadas essencialmente em técnicas de programação dinâmica e algoritmos de *Branch-and-Bound*. Em [Abdul Razaq *et al.*, 1990] é apresentado um estudo comparativo de alguns destes métodos. Em [Potts e Wassenhove, 1991] são referidas algumas

tentativas de aplicação de regras de prioridade (EDD, SWPT) na resolução deste problema, as quais não permitiram a obtenção de soluções de boa qualidade. Outros trabalhos podem ser encontrados em [Crauwels *et al.*, 1998], [Congram *et al.*, 1988], [Madureira, 1999] e [Madureira *et al.*, 2001].

Uma vez que os problemas SMTWT são do tipo *NP-hard*, os algoritmos capazes de encontrarem a solução óptima para este tipo de problemas necessitam de um tempo computacional que aumenta exponencialmente com o tamanho do problema [Madureira, 1999], [Morton *et al.*, 1993], [Baker, 1974], [Lawler, 1977], [Abdul Razaq *et al.*, 1990] e [Potts e Wassenhove, 1991]. Muitos algoritmos foram desenvolvidos para resolver este tipo de problemas, como por exemplo: os algoritmos *branch and bound* [Joel, 1972], [Rinnooy *et al.*, 1975], [Potts *et al.*, 1985] e os algoritmos de programação dinâmica [Schrage *et al.*, 1978]. Estes, foram desenvolvidos para gerar soluções exactas, isto é, soluções óptimas. Mas, os algoritmos *branch and bound* são limitados pelo tempo computacional que exigem e os algoritmos de programação dinâmica são limitados pelo armazenamento computacional que necessitam, especialmente quando o número de *jobs* é superior a 50. Recentemente, os problemas SMTWT têm sido estudados com heurísticas. Estas heurísticas geram boas ou mesmo soluções óptimas, mas nem sempre garantem soluções óptimas. Nos últimos anos, foram propostas algumas heurísticas para a resolução dos problemas SMTWT, nomeadamente, *Simulated Annealing*, *Tabu Search* e *Ant Colony* [Peter *et al.*, 1997], [Crauwels *et al.*, 1998], [Madureira, 1999] e [Morton *et al.*, 1993].

Exemplo 6.1

Considere-se o problema de sequenciamento de 5 *jobs* numa só máquina. Os tempos de processamento dos 5 *jobs* são respectivamente p_1=2, p_2=4, p_3=1, p_4=3 e p_5=3, as datas de entrega d_1=5, d_2=7, d_3=11, d_4=9 e d_5=8, e as penalizações dos *jobs* por cada unidade de tempo em atraso w_1=1, w_2= 6, w_3=2, w_4=3 e w_5=2.

A tabela 6.1 e o diagrama de *Gantt* apresentado na figura 6.23 correspondem à sequência de *jobs* $\{J_3, J_1, J_2, J_5, J_4\}$. Com base na tabela 6.1 e na figura 6.23 podemos concluir que o tempo de conclusão dos *jobs* é igual a 13 períodos de tempo, e que os *jobs* 4 e 5 se atrasaram em 4 e 2 períodos de tempo, respectivamente. Considerando as penalizações w_i (*weight*) atribuídas aos *jobs* podemos determinar para esta sequência o valor da função que minimiza a soma dos atrasos pesados (*weighted tardiness*) $\sum w_i T_i = 16$, como se pode observar na tabela 6.1.

Tabela 6.1 Sequência dos *jobs* $\{J_3, J_1, J_2, J_5, J_4\}$

Job	p_i	d_i	w_i	r_i	C_i	L_i	T_i	w_iT_i
3	1	11	2	0	1	-10	0	0
1	2	5	1	0	3	-2	0	0
2	4	7	6	0	7	0	0	0
5	3	8	2	0	10	2	2	4
4	3	9	3	0	13	4	4	12
							$\sum w_iT_i$	**16**

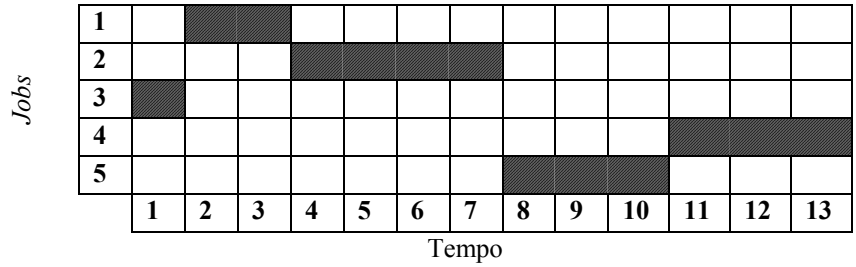


Figura 6.23 Diagrama de *Gantt*.

Como sabemos, para qualquer problema de sequenciamento o número de sequências possíveis é obtido por $(n!)^m$. Desta forma, para o exemplo em questão existem 120 sequências possíveis. Assim, seria necessário analisarmos as restantes sequências para determinarmos a sequência óptima que minimiza a soma dos atrasos pesados $min\sum w_iT_i$.

6.5 Testes computacionais

Esta secção apresenta os resultados dos testes computacionais realizados com o objectivo de examinar e avaliar os operadores genéticos apresentados na secção 6.2. Os testes foram realizados num computador *Pentium* IV, 3 GHz e 1 GB de RAM [Ferrolho e Crisóstomo, 2006a], [Ferrolho e Crisóstomo, 2006d] e [Ferrolho e Crisóstomo, 2007b].

6.5.1 Condições utilizadas nos testes computacionais

A tabela 6.2 apresenta a instância de 40 *jobs*, gerada aleatoriamente, tendo sido utilizada nos testes computacionais. Para cada *job i (i=1,...,n)* o tempo de processamento p_i é um número inteiro escolhido aleatoriamente no intervalo [1, 100], e o peso do atraso w_i (*weighted tardiness*) é um número inteiro pertencente ao intervalo [1, 10]. As datas de entrega d_i para cada *job* são números inteiros obtidos aleatoriamente do intervalo [*P*(1-*TF*-*RDD*/2), *P*(1-*TF*+*RDD*/2)], onde *RDD* representa o intervalo das datas de entrega (*range of due dates*) e *TF* representa o factor de atraso (*tardiness factor*), podendo ambos assumir valores iguais a 0,2; 0,4; 0,6; 0,8 ou 1,0.

Variando os parâmetros *RDD* e *TF* é possível gerar instâncias de problemas com diferentes graus de dificuldade. Por último, *P* representa a soma dos tempos de processamento de todos os *jobs*, isto é, $P = \sum_{i=1}^{n} p_i$.

A tabela A1 do apêndice A mostra a população inicial (Ψ_t) gerada aleatoriamente e utilizada nos testes computacionais. Esta é constituída por 100 cromossomas (N_{pop}=100) e cada cromossoma possui 40 *jobs* (n=40). Cada cromossoma representa uma possível solução para o problema, não podendo ter *jobs* repetidos ou ausentes.

Refira-se que, por vezes, é difícil comparar o desempenho dos diferentes métodos de optimização combinatória, pois os autores usam problemas (instâncias) gerados por ferramentas criadas para o efeito, onde os tempos de processamento e as datas de entrega dos *jobs* são gerados aleatoriamente. Uma vez que as instâncias dos problemas usados nos testes computacionais não são publicadas, raramente voltam a ser usadas por outros autores. São excepções os problemas teste encontrados em [Fisher e Thompson, 1963] e [Lawrence, 1984] vulgarmente usados por vários autores. No entanto, é de referir a existência de um esforço na criação de bibliotecas de problemas teste, como por exemplo os disponibilizados em [http#4]. De forma a contrariarmos, também, esta tendência, a tabela A1 do apêndice A apresenta a população inicial utilizada nos testes computacionais e a tabela 6.2 a instância utilizada nos mesmos testes.

Tabela 6.2 Instância utilizada nos testes computacionais

WT40	***Jobs***	1	2	3	4	5	6	7	8	9	10
	p_i	71	58	89	62	8	31	74	52	71	85
	d_i	1419	1682	1683	1617	1703	1549	1741	1634	1580	1588
	w_i	6	3	9	3	8	8	3	8	10	4
	r_i	0	0	0	0	0	0	0	0	0	0
	Jobs	11	12	13	14	15	16	17	18	19	20
	p_i	1	77	35	30	96	12	4	29	64	34
	d_i	1694	1574	1548	1730	1535	1438	1501	1504	1587	1687
	w_i	7	8	10	5	5	6	5	10	3	6
	r_i	0	0	0	0	0	0	0	0	0	0
	Jobs	21	22	23	24	25	26	27	28	29	30
	p_i	8	98	86	22	6	6	24	61	86	76
	d_i	1472	1507	1389	1454	1404	1522	1526	1681	1506	1584
	w_i	9	4	3	2	7	10	6	10	9	3
	r_i	0	0	0	0	0	0	0	0	0	0
	Jobs	31	32	33	34	35	36	37	38	39	40
	p_i	17	36	63	83	81	37	80	56	11	57
	d_i	1720	1767	1621	1677	1487	1513	1591	1620	1771	1712
	w_i	2	6	6	9	10	9	7	5	5	6
	r_i	0	0	0	0	0	0	0	0	0	0

6.5.2 Avaliação dos operadores de cruzamento

Foi utilizado o HybFlexGA apresentado na secção 6.3 com o objectivo de se examinarem e se avaliarem os doze operadores de cruzamento apresentados na secção 6.2. Cada operador de cruzamento foi examinado e avaliado com base nas seguintes condições:

- Número de testes: 20;
- População inicial (Ψ_t): constante e gerada aleatoriamente (ver tabela A1 do apêndice A);
- Número de *jobs*: 40;
- Instância usada: constante (ver tabela 6.2);
- Tamanho da população (N_{pop}): 20, 40, 60, 80 e 100 cromossomas;
- Condição de paragem: 1000 gerações;
- Probabilidade de cruzamento (P_c): 0,2, 0,4, 0,6, 0,8 e 1,0;
- Operador de mutação: não foi utilizado;
- Probabilidade de mutação (P_m): não foi utilizada.

À semelhança de outros autores [Manderick e Spiessens, 1994] e [Murata *et al.*, 1996], o operador de mutação e a probabilidade de mutação não foram utilizados nos testes computacionais (*Mutation operator*: *null* e P_m=0), porque a finalidade destes é avaliar o desempenho dos operadores de cruzamento. Cada operador de cruzamento foi analisado para diferentes tamanhos de população (N_{pop}=20, 40, 60, 80 e 100) e para diferentes probabilidades de cruzamentos (P_c=0,2, 0,4, 0,6, 0,8 e 1,0). Para cada uma destas combinações, por exemplo: *one--point crossover: 1 child*, N_{pop}=20 e P_c=0,2, foram realizados 20 testes, partindo sempre da mesma população inicial.

Com o objectivo de avaliar o desempenho de cada operador de cruzamento, foi utilizada a seguinte medida de desempenho:

$$Performance = f(\bar{x}_{inicial}) - f(\bar{x}_{final}), \tag{6.1}$$

onde $x_{inicial}$ é o melhor cromossoma da população inicial e x_{final} o melhor cromossoma da população final. Assim, $f(\bar{x}_{inicial})$ é a média do *fitness* (dos 20 testes realizados) dos melhores cromossomas da população inicial e $f(\bar{x}_{final})$ a média do *fitness* dos melhores cromossomas no final das 1000 gerações. A tabela 6.3 apresenta a performance, dos 20 testes realizados, obtida para cada um dos operadores de cruzamento, para os diferentes tamanhos da população e probabilidades de cruzamento. Nesta tabela, a negrito (*bold*) e a amarelo encontram-se os melhores valores de performance obtidos nos vários testes realizados.

Tabela 6.3 Performance dos vários operadores de cruzamento

		Número de *Jobs*: 40				
Tamanho da população (N_{pop})		**20**	**40**	**60**	**80**	**100**
Operador de cruzamento	P_c	**Performance**	**Performance**	**Performance**	**Performance**	**Performance**
One-point crossover: 1 child	0,2	2873,40	3114,25	3112,05	3181,65	3479,45
	0,4	3036,25	2993,50	3145,70	3398,35	3388,30
	0,6	3005,65	3187,80	3179,30	3354,10	**3570,50**
	0,8	3020,10	3159,20	3335,60	3350,00	3560,10
	1,0	**3145,5**	**3306,60**	**3393,65**	**3495,75**	3547,20
Two-point crossover: 1 child (Ver. I)	0,2	2958,45	3156,10	3416,95	3517,90	3508,85
	0,4	3137,70	3350,15	3453,50	3467,75	3599,75
	0,6	3153,80	3420,55	3404,20	3370,60	3523,85
	0,8	3209,35	3437,70	3513,45	**3548,15**	3622,00
	1,0	**3320,45**	**3496,80**	**3571,70**	3524,35	**3624,65**
Two-point crossover: 1 child (Ver. II)	0,2	3183,60	3440,20	3643,30	3749,50	3764,40
	0,4	3250,95	3556,25	3677,65	3752,30	3774,15
	0,6	3385,05	3622,20	**3772,20**	**3785,25**	3754,85
	0,8	3465,25	**3659,85**	3717,80	3781,70	**3789,30**
	1,0	**3524,5**	3599,65	3760,20	3728,95	3768,20
One-point crossover: 2 children	0,2	2857,80	2927,40	3032,15	3435,50	3490,00
	0,4	3073,80	3185,80	3252,85	3432,95	3523,05
	0,6	3205,20	3299,10	3336,85	3516,80	**3647,75**
	0,8	3214,60	3320,15	3421,55	3556,55	3563,05
	1,0	**3315,2**	**3361,10**	**3433,10**	**3593,70**	3513,45
Two-point crossover: 2 children (Ver. I)	0,2	3142,85	3335,85	3372,25	3370,65	3559,70
	0,4	3112,70	3353,35	3465,00	3567,15	3609,25
	0,6	3377,85	3398,85	**3616,05**	3541,70	3576,30
	0,8	3475,50	3462,30	3575,80	3623,10	3609,55
	1,0	**3548,5**	**3482,55**	3525,45	**3662,05**	**3643,00**
Two-point crossover: 2 children (Ver. II)	0,2	3389,25	3671,15	3651,70	3675,55	3802,40
	0,4	3438,35	3583,15	3734,75	3771,30	3775,95
	0,6	3576,95	3660,60	3734,20	3781,10	3785,95
	0,8	3581,05	3648,00	**3769,00**	**3808,45**	3788,30
	1,0	**3660,05**	**3696,05**	3754,80	3789,00	**3821,75**
Two-point crossover: 3 children (Ver. I)	0,2	3178,95	3385,30	3531,50	3475,20	3628,65
	0,4	33640	3363,70	3500,05	3541,85	3667,00
	0,6	**3538,65**	3499,25	3555,00	3503,10	3639,50
	0,8	3392,00	3494,95	**3566,90**	**3680,20**	**3676,10**
	1,0	3508,55	**3521,65**	3560,65	3636,15	3667,70
Two-point crossover: 3 children (Ver. II)	0,2	3447,50	3624,10	3622,15	3744,60	3785,35
	0,4	3579,25	3727,70	3755,00	3788,30	3797,60
	0,6	**3693,05**	3733,10	3763,90	3783,45	3799,05
	0,8	3643,70	3758,80	**3799,00**	3790,35	3791,00
	1,0	3615,00	**3768,55**	3787,90	**3819,65**	**3822,95**
Two-point crossover: 4 children	0,2	3531,25	3657,55	3726,65	3775,20	3796,85
	0,4	3591,05	3734,55	3742,30	3820,25	3828,15
	0,6	**3710,65**	3715,40	3775,35	3793,50	3796,05
	0,8	3677,25	**3755,60**	3798,95	3822,75	3819,85
	1,0	3705,45	3747,50	**3803,75**	**3828,65**	**3834,05**
Order crossover	0,2	2807,50	3268,05	3302,20	3357,00	3433,55
	0,4	3071,45	3297,10	3399,55	3460,60	3545,70
	0,6	3272,60	3404,80	3523,85	3550,55	3588,00
	0,8	3352,30	3430,10	**3547,50**	3597,45	3631,10
	1,0	**3363,55**	**3594,25**	3546,85	**3624,10**	**3635,40**
Cycle crossover	0,2	3187,10	3342,95	3489,55	3555,45	3627,70
	0,4	3457,60	3525,30	3608,00	3693,10	3674,45
	0,6	3524,55	3656,00	3671,30	3722,45	3718,95
	0,8	3576,65	3694,95	**3729,75**	**3788,65**	3753,85
	1,0	**3671,80**	**3703,25**	3728,85	3772,90	**3758,40**
Position based crossover	0,2	2990,70	3260,05	3327,35	3395,60	3470,95
	0,4	3389,60	3506,35	3589,00	3676,40	3727,85
	0,6	3512,25	3681,05	3665,80	3761,90	3775,95
	0,8	3572,85	3726,85	3757,55	3770,65	**3801,10**
	1,0	**3687,85**	**3740,75**	**3779,65**	**3805,80**	3794,85

Em todos os testes computacionais realizados, utilizamos sempre a mesma população inicial (tabela A1 do apêndice A). Mas, o tamanho da população N_{pop} variou consoante o caso em estudo, isto é, para N_{pop}=20 foram utilizados somente os primeiros 20 cromossomas da população inicial, para N_{pop}=40 foram utilizados os primeiros 40 cromossomas da população inicial, e assim sucessivamente.

Os resultados obtidos na tabela 6.3 mostram que o tamanho da população influencia o desempenho do operador de cruzamento. Verificamos que, com o aumento do tamanho da população, há uma melhoria da performance dos vários operadores de cruzamento. Esta melhoria é conseguida com um custo computacional maior, uma vez que o algoritmo efectua o processamento duma população maior. Podemos concluir, dos resultados obtidos na tabela 6.3, que os operadores de cruzamento têm um melhor desempenho para probabilidades de cruzamento (P_c) pertencentes ao intervalo [0,6; 1,0], isto é, $P_c \in [0{,}6; 1{,}0]$. De notar que os três operadores de cruzamento com melhor performance apresentam probabilidades de cruzamento de 1,0.

Os resultados da tabela 6.3 mostram ainda que, normalmente, a performance dos operadores de cruzamento aumenta com o tamanho da população. Porém, há situações pontuais em que o aumento da população não se traduz numa melhoria da performance. Por exemplo, verificamos que os operadores de cruzamento *two-point crossover: 2 children (Ver. I)*, *two-point crossover: 3 children (Ver. I)*, *cycle crossover* e *position based crossover* obtiveram uma diminuição de performance com o aumento do tamanho da população de 80 para 100 cromossomas.

Na tabela 6.4 encontra-se a grelha de classificação, por ordem decrescente, dos operadores de cruzamento utilizados nos testes. Da leitura da tabela 6.4 verificamos que:

- A melhor performance foi obtida para o operador *two-point crossover: 4 children* com P_c=1,0 e N_{pop}=100. Para este caso a média da performance foi de 3834,05;
- A pior performance foi obtida para o operador *one-point crossover: 1 child* com P_c =0,6 e N_{pop} =100. Para este caso a média da performance foi de 3570,5.

Tabela 6.4 Classificação dos operadores de cruzamento

Posição	Operador de cruzamento	P_c	N_{pop}	Performance
1º	Two-point crossover: 4 children	1,0	100	3834,05
2º	Two-point crossover: 3 children (Ver. II)	1,0	100	3822,95
3º	Two-point crossover: 2 children (Ver. II)	1,0	100	3821,75
4º	Position based crossover	1,0	80	3805,80
5º	Two-point crossover: 1 child (Ver. II)	0,8	100	3789,30
6º	Cycle crossover	0,8	80	3788,65
7º	Two-point crossover: 3 children (Ver. I)	0,8	80	3680,20
8º	Two-point crossover: 2 children (Ver. I)	1,0	80	3662,05
9º	One-point crossover: 2 children	0,6	100	3647,75
10º	Order crossover	1,0	100	3635,40
11º	Two-point crossover: 1 child (Ver. I)	1,0	100	3624,65
12º	One-point crossover: 1 child	0,6	100	3570,50

Os gráficos apresentados na figura 6.24 mostram a média do *fitness* dos seis melhores e dos seis piores operadores de cruzamento ao longo das 1000 gerações. O objectivo da ampliação dos gráficos (figura 6.24(b) e (d)) é permitir observar melhor o desempenho destes operadores, entre a geração 500 e 1000.

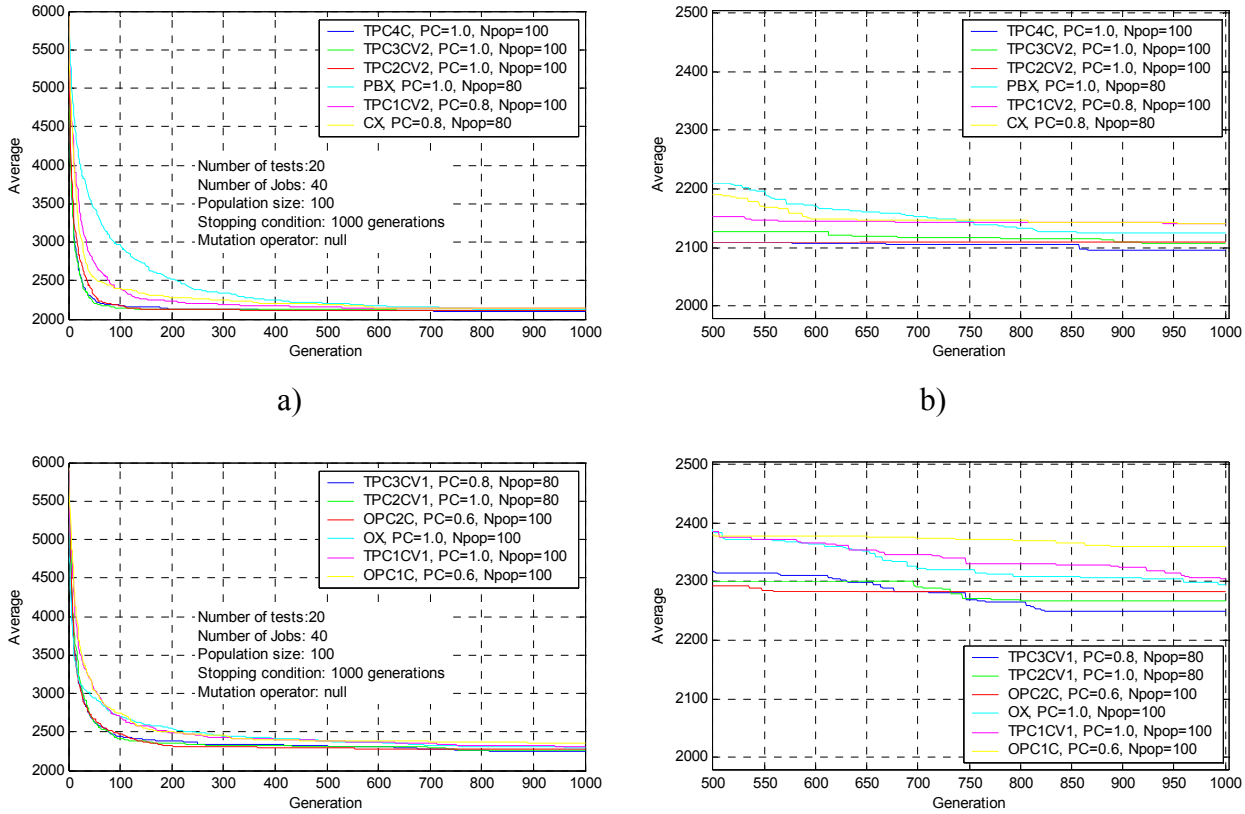
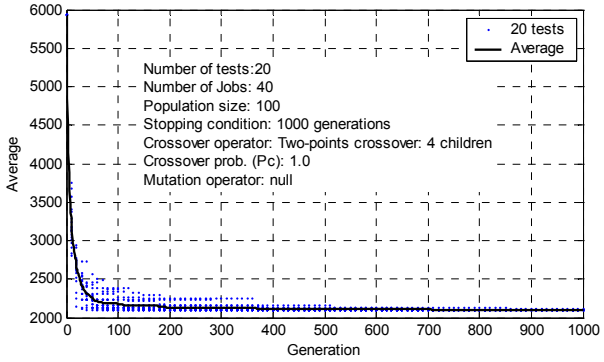


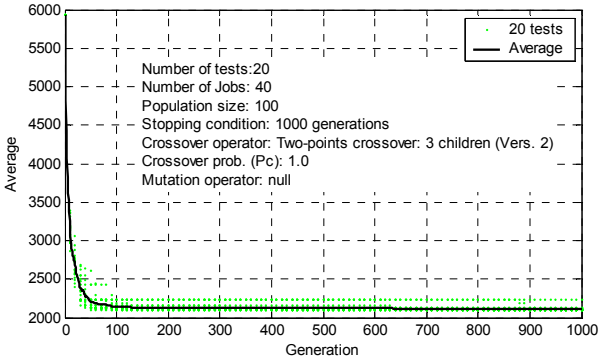
Figura 6.24 Comparação dos operadores de cruzamento:

a) Os seis melhores (da geração 0 à 1000)
b) Os seis melhores (da geração 500 à 1000)
c) Os seis piores (da geração 0 à 1000)
d) Os seis piores (da geração 500 à 1000)

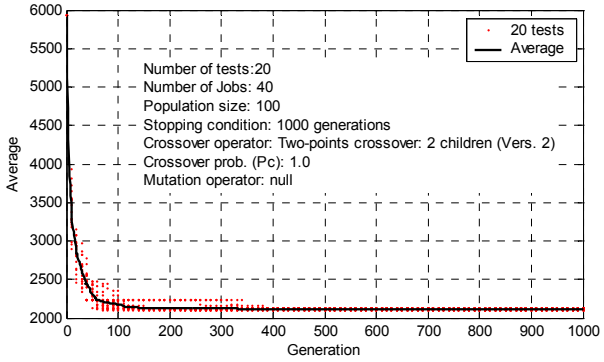
Dos 12 operadores de cruzamento utilizados nos testes, seleccionamos os 6 melhores para uma análise mais detalhada. Assim, para cada operador de cruzamento, o gráfico apresentado na figura 6.25 mostra os resultados obtidos nos 20 testes (os pontos nos gráficos) e a respectiva média (as linhas nos gráficos). Assim, os pontos representam os resultados obtidos, de 10 em 10 gerações, nos 20 testes realizados. Os três primeiros operadores de cruzamento apresentam uma evolução muito rápida nas primeiras 100 gerações e uma evolução mais lenta da geração 100 à 1000. Os restantes três operadores de cruzamento (*position based crossover*, t*wo-point crossover: 1 child (Ver. II)* e *cycle crossover)* apresentam uma evolução mais lenta relativamente aos três primeiros. O operador *position based crossover* apesar de ter uma evolução muito lenta nas primeiras 500 gerações consegue recuperar essa desvantagem nas últimas 500 gerações, ficando desta forma posicionado na tabela de classificação em 4º lugar.



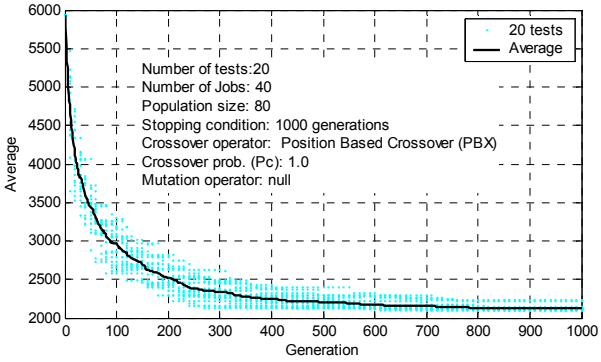
a)



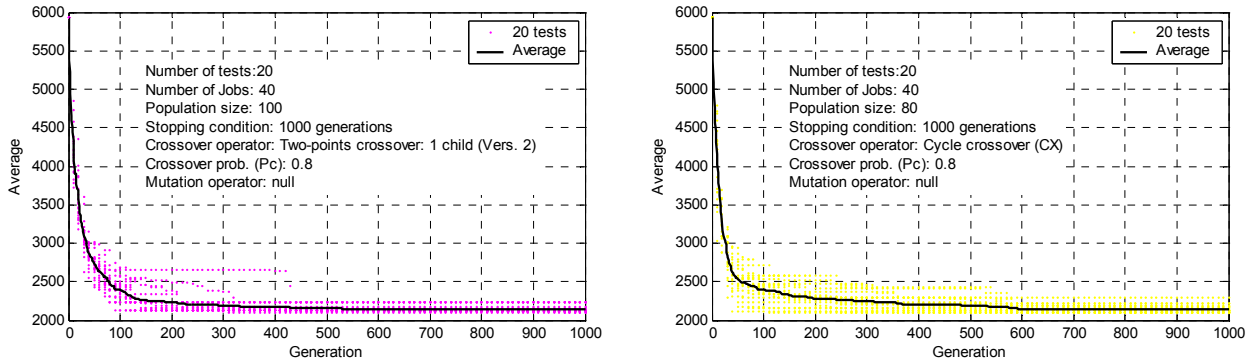
b)



c)



d)



e) f)

Figura 6.25 Os seis melhores operadores de cruzamento:
a) *Two-point crossover: 4 children* b) *Two-point crossover: 3 children (Ver. II)*
c) *Two-point crossover: 2 children (Ver. II)* d) *Position based crossover*
e) *Two-point crossover: 1 child (Ver. II)* f) *Cycle crossover*

O gráfico da figura 6.26 permite uma visualização simultânea dos seis melhores operadores de cruzamento. Neste gráfico mantemos, para cada operador de cruzamento, as cores usadas nos gráficos da figura 6.25. Para os três melhores operadores de cruzamento, verificamos que a dispersão dos valores relativamente à média, dos 20 testes realizados, aumenta gradualmente até à geração número 100 e a partir daí vai diminuindo gradualmente. Da leitura dos gráficos, também verificamos que a dispersão dos valores relativamente à média é maior nos operadores de cruzamento *position based crossover*, t*wo-point crossover: 1 child (Ver. II)* e *cycle crossover*.

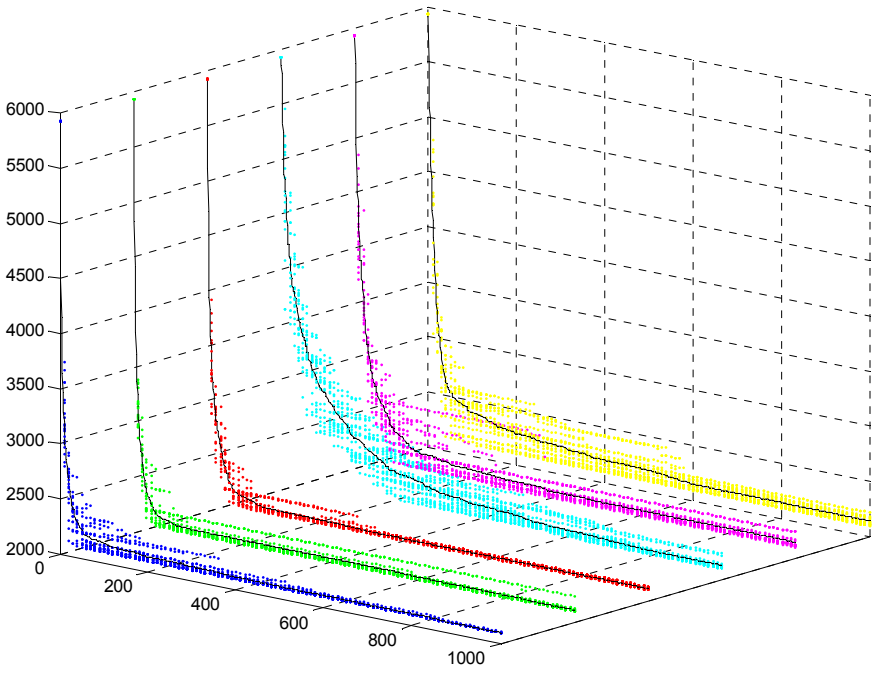


Figura 6.26 Os seis melhores operadores de cruzamento.

O gráfico mostrado na figura 6.27 apresenta a média do tempo do CPU, em segundos, dos seis melhores operadores de cruzamento obtidos nos testes computacionais. Da leitura do gráfico, observamos que, para os seis operadores de cruzamento, o custo computacional aumenta linearmente com o aumento do número de gerações. Verificamos, também, que o operador *two--point crossover: 4 children* é o mais exigente do ponto de vista computacional e que o *cycle crossover* é o menos exigente. Na verdade, os operadores de cruzamento *position based crossover* e *cycle crossover* são os menos exigentes do ponto de vista computacional. Tal facto deve-se a estes operadores não utilizarem pontos de corte no processo de geração de descendentes, tornando, desta forma, o algoritmo menos exigente computacionalmente.

Para o mesmo número de gerações, o *position based crossover* e *cycle crossover* precisam de menos tempo de CPU (ver figura 6.27), no entanto, estes operadores de cruzamento apresentam a pior média de fitness (ver figura 6.26).

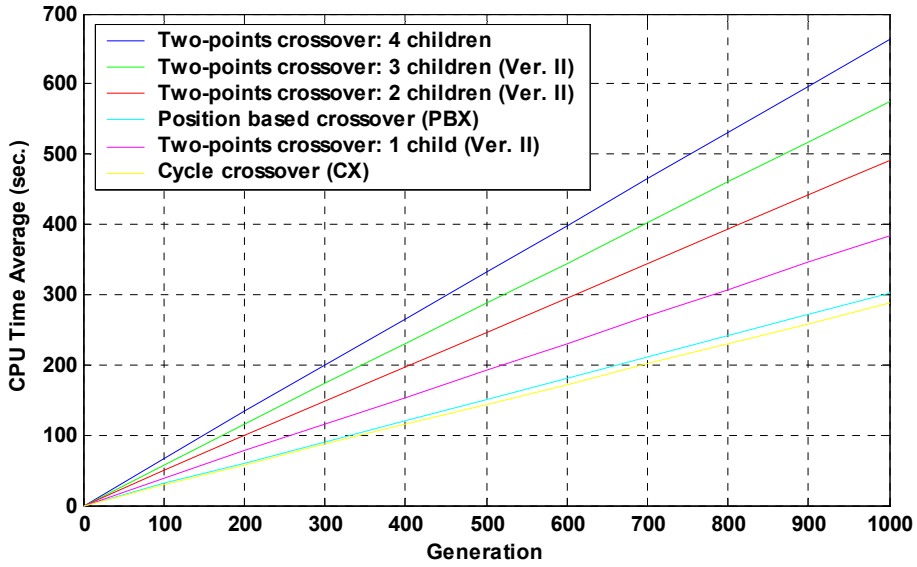
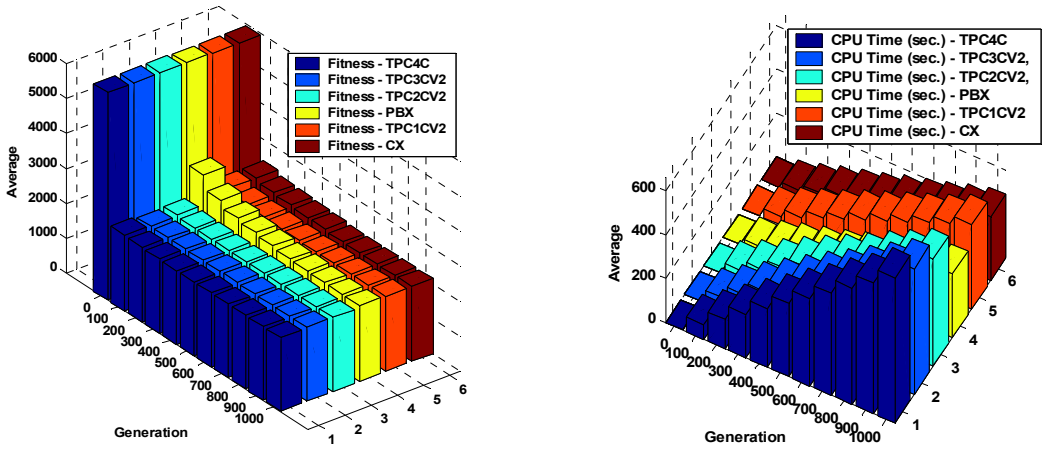


Figura 6.27 Média do tempo do CPU.

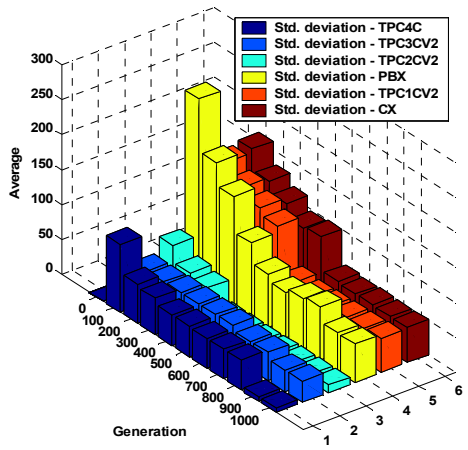
Os três gráficos da figura 6.28, correspondentes aos seis melhores operadores de cruzamento, apresentam a média do *fitness*, a média do tempo do CPU (em segundos) e a média do desvio padrão obtidas nos testes computacionais realizados. Em todos os gráficos, as barras representam os resultados obtidos nos testes computacionais de 100 em 100 gerações. A figura 6.28(a) mostra que os três melhores operadores de cruzamento revelam uma evolução muito rápida nas primeiras 100 gerações mas, estes operadores também necessitam de mais tempo de CPU (ver figura 6.28(b)). Por outro lado, a figura 6.28(c) mostra que estes operadores de cruzamento, ao longo das 1000 gerações, apresentam um valor médio de desvio padrão mais baixo relativamente

aos outros operadores de cruzamento. Isto significa que a dispersão dos valores obtidos (nos 20 testes computacionais) relativamente à média do *fitness* é mais pequena nestes três operadores de cruzamento (*two-point crossover: 4 children*, *two-point crossover: 3 children (Ver. II)*, *two-point crossover: 2 children (Ver. II)*).

Os operadores de cruzamento *position based crossover*, *two-point crossover: 1 child (Ver. II)* e *cycle crossover* precisam de menos tempo de CPU relativamente aos três primeiros operadores de cruzamento (ver figura 6.28(b)). Contudo, como mostram as figuras 6.28(a) e 6.28(c), estes operadores apresentam uma evolução mais lenta e um desvio padrão maior ao longo das 1000 gerações, relativamente aos operadores *two-point crossover: 4 children*, *two-point crossover: 3 children (Ver. II)*, *two-point crossover: 2 children (Ver. II)* [Ferrolho *et al.*, 2006a] e [Ferrolho *et al.*, 2006b].



a) b)



c)

Figura 6.28 Os seis melhores operadores de cruzamento:
a) Média do *fitness* b) Média do CPU (seg.) c) Média do desvio padrão

6.5.3 Avaliação dos operadores de mutação

À semelhança dos testes computacionais realizados na avaliação dos operadores de cruzamento, nestes testes também não utilizámos o operador de cruzamento e a probabilidade de cruzamento (*Crossover operator: null* e P_c=0). Cada operador de mutação foi analisado para diferentes tamanhos de população (N_{pop}=20, 40, 60, 80 e 100) e para diferentes probabilidades de mutação (P_m=0,2, 0,4, 0,6, 0,8 e 1,0). Nestes testes computacionais utilizámos a mesma população inicial usada nos testes computacionais realizados na avaliação dos operadores de cruzamento (tabela A1 do apêndice A).

Com o objectivo de avaliar o desempenho de cada operador de mutação, utilizámos a medida de desempenho (6.1), usada na avaliação dos operadores de cruzamento. A tabela 6.5 apresenta a performance, dos 20 testes realizados, obtida para cada um dos operadores de mutação, para os diferentes tamanhos da população e probabilidades de mutação. Nesta tabela, a negrito (*bold*) e a amarelo encontram-se os melhores valores de performance obtidos nos vários testes realizados.

Tabela 6.5 Performance dos vários operadores de mutação

		Número de *Jobs*: 40				
Tamanho da população (N_{pop})		**20**	**40**	**60**	**80**	**100**
Operador de mutação	**P_m**	**Performance**	**Performance**	**Performance**	**Performance**	**Performance**
Adjacent two-job change	0,2	**2846,30**	2953,00	3108,70	3079,95	3159,00
	0,4	2778,90	2952,35	2985,30	3122,65	**3250,35**
	0,6	2808,75	2954,85	3048,80	3079,40	3209,25
	0,8	2756,70	**3107,75**	**3140,10**	3120,60	3204,50
	1,0	2755,00	2884,05	3091,30	**3136,90**	3201,45
Arbitrary two-job change	0,2	3490,20	3615,45	3745,20	3723,50	3736,30
	0,4	3717,75	3798,20	3797,50	3803,00	3802,75
	0,6	3779,10	3799,10	**3821,90**	3816,90	3813,40
	0,8	3783,00	3805,95	3817,00	**3818,10**	**3826,35**
	1,0	**3797,75**	**3811,40**	3821,15	3815,45	3822,60
Arbitrary three-job change	0,2	3579,25	3681,75	3631,35	3687,35	3729,10
	0,4	3661,30	3727,15	3780,35	3793,70	3794,25
	0,6	3704,75	3787,25	3773,75	3796,10	**3808,30**
	0,8	3710,85	3770,00	3788,65	3810,25	3808,05
	1,0	**3772,65**	**3800,15**	**3814,90**	**3810,95**	3802,10
Shift change	0,2	2863,40	3150,55	3246,70	3319,65	3356,65
	0,4	3140,70	3300,55	3343,45	3336,15	3378,40
	0,6	3297,95	3463,75	3450,40	3445,30	3464,75
	0,8	3202,05	**3484,75**	**3673,45**	3485,10	**3586,35**
	1,0	**3385,85**	3369,75	3590,60	**3599,45**	3581,45
Arbitrary 20%-job change	0,2	3631,20	3756,30	3743,10	3791,00	3786,30
	0,4	3779,15	3799,50	3798,95	3806,20	3820,75
	0,6	3804,30	3810,50	3814,60	3825,10	3829,35
	0,8	3808,85	**3822,65**	3829,20	3830,85	3823,65
	1,0	**3811,55**	3821,75	**3830,00**	**3832,90**	**3833,95**

A tabela 6.6 apresenta a grelha de classificação, por ordem decrescente, dos operadores de mutação utilizados nos testes. A melhor performance foi obtida para o operador *arbitrary 20%--job change*, com uma probabilidade de mutação P_m=1,0 e uma população N_{pop}=100.

Tabela 6.6 Classificação dos operadores de mutação

Posição	Operador de mutação	P_m	N_{pop}	Performance
1º	Arbitrary 20%-job change	1,0	100	3833,95
2º	Arbitrary two-job change	0,8	100	3826,35
3º	Arbitrary three-job change	1,0	60	3814,90
4º	Shift change	0,8	60	3673,45
5º	Adjacent two-job change	0,4	100	3250,35

A figura 6.29 mostra o desempenho dos operadores de mutação ao longo das 1000 gerações. Para cada um destes operadores de mutação foram realizados 20 testes computacionais e a condição de paragem utilizada foi, mais uma vez, 1000 gerações. Da leitura do gráfico verificamos que os operadores *arbitrary 20%-job change*, *arbitrary two-job change* e *arbitrary three-job change* têm um desempenho muito próximo a partir da geração 500. No entanto, mesmo na geração 1000, há ligeiras diferenças na performance deles. Estes três operadores apresentam uma maior divergência até à geração 200, sendo o *arbitrary 20%-job change* o que apresenta maior desempenho. O *adjacent two-job change* destaca-se pelo seu fraco desempenho relativamente aos restantes operadores de mutação.

Se não considerarmos o *adjacent two-job change*, pelo facto deste ocupar a última posição na tabela de classificação (tabela 6.6), podemos afirmar que os operadores de mutação têm um melhor desempenho para probabilidades de mutação (P_m) pertencentes ao intervalo [0,8; 1,0]. A performance dos operadores de mutação, à semelhança do que aconteceu nos operadores de cruzamento, varia com o aumento do tamanho da população. Contudo, e como mostra a tabela 6.6, os melhores resultados foram obtidos para os tamanhos de população de 60 e 100 cromossomas. Assim, e com a finalidade de estabelecermos um intervalo para o tamanho da população, podemos afirmar que é vantajoso usarmos tamanhos de populações pertencentes ao intervalo [60, 100], isto é, $N_{pop} \in [60, 100]$.

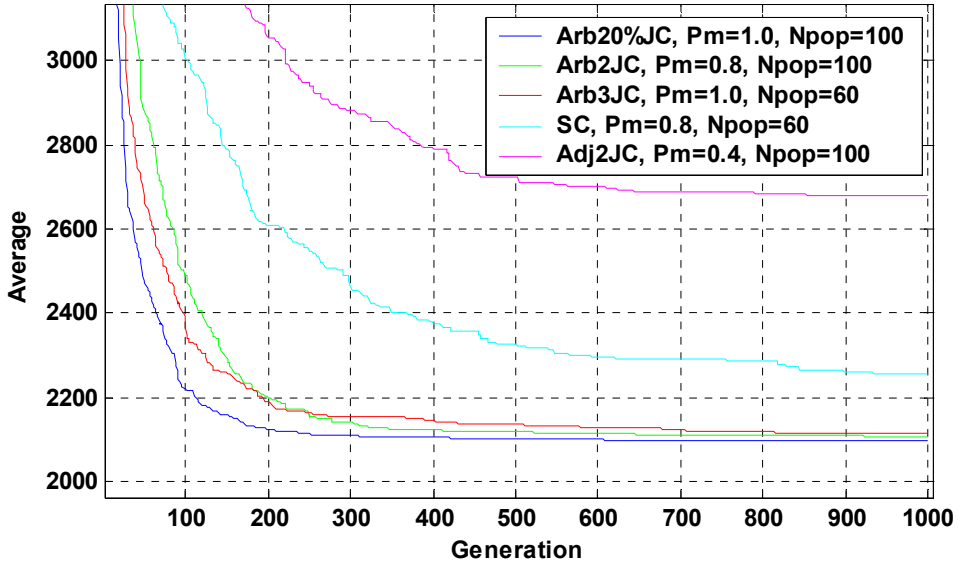


Figura 6.29 Comparação do desempenho dos operadores de mutação.

Da leitura do gráfico da figura 6.29 e da tabela 6.6, verificamos que o operador *arbitrary 20%--job change* é o que apresenta melhor desempenho ao longo das 1000 gerações e, também, a melhor performance (obtida através da equação (6.1)). Assim, face ao desempenho deste operador, surgiu a seguinte questão: se variarmos a percentagem de *jobs*, como é que se comporta este operador? Para respondermos a esta questão, foram realizados testes computacionais para 10% e 30% de *jobs*. O gráfico da figura 6.30 e a tabela 6.7 mostram o desempenho e a performance deste operador, para as diferentes percentagens. Assim, e face aos resultados obtidos, podemos concluir que é mais vantajoso utilizar a percentagem de 20% neste operador de mutação.

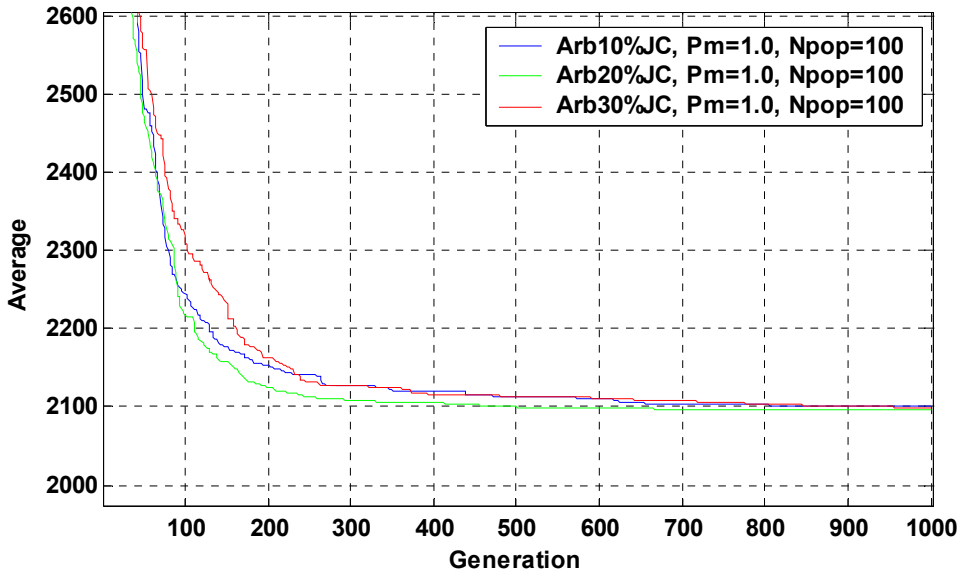


Figura 6.30 Desempenho para as diferentes percentagens.

Tabela 6.7 Performance para as diferentes percentagens.

Operador de mutação	P_m	N_{pop}	Performance
Arbitrary 10%-job change	1,0	100	3829,20
Arbitrary 20%-job change	1,0	100	3833,95
Arbitrary 30%-job change	1,0	100	3830,65

Dos cinco operadores de mutação utilizados nos testes computacionais, seleccionamos os três melhores para uma análise mais detalhada. Assim, para cada operador de mutação, o gráfico apresentado na figura 6.31 mostra os resultados obtidos, de 10 em 10 gerações, nos 20 testes computacionais realizados (representados nos gráficos por pontos) e a respectiva média obtida (representada nos gráficos por uma linha).

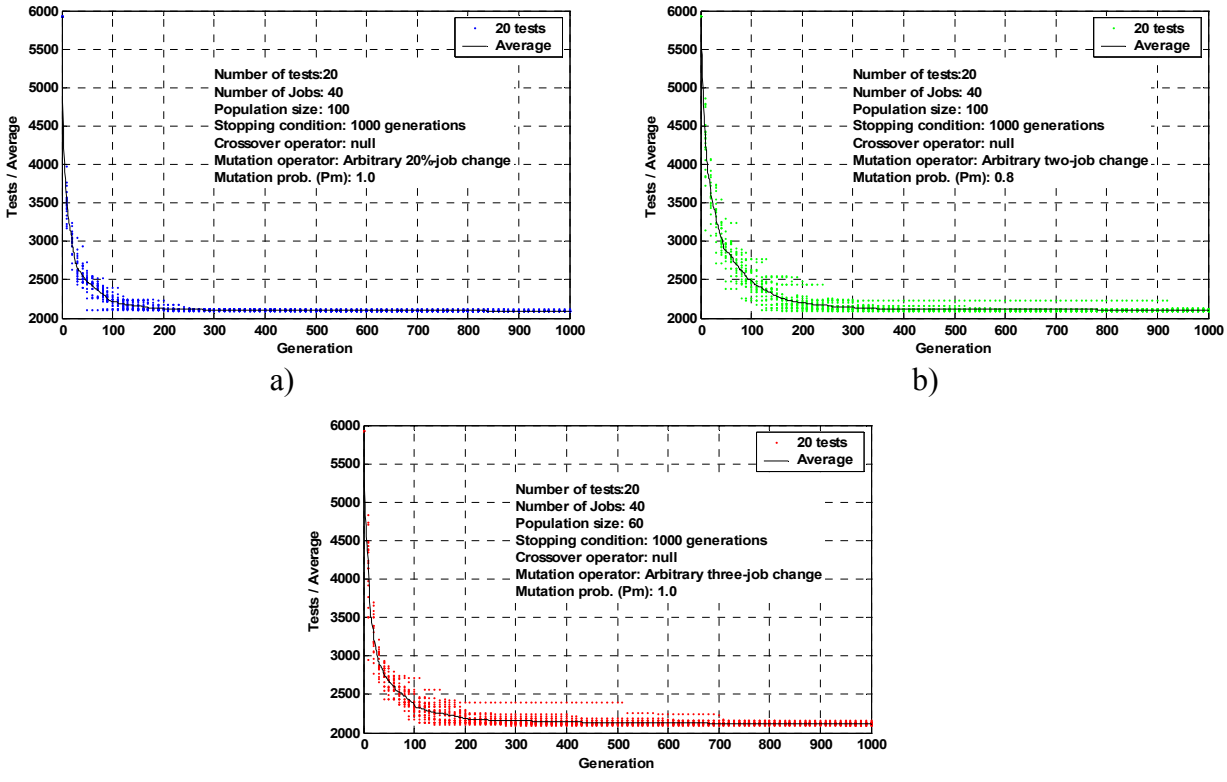


Figura 6.31 Os três melhores operadores de mutação:

a) *Arbitrary 20%-job change* b) *Arbitrary two-job change* c) *Arbitrary three-job change*

O gráfico da figura 6.32 permite uma visualização simultânea dos três melhores operadores de mutação. Neste gráfico mantemos, para cada operador de mutação, as cores usadas nos gráficos da figura 6.31.

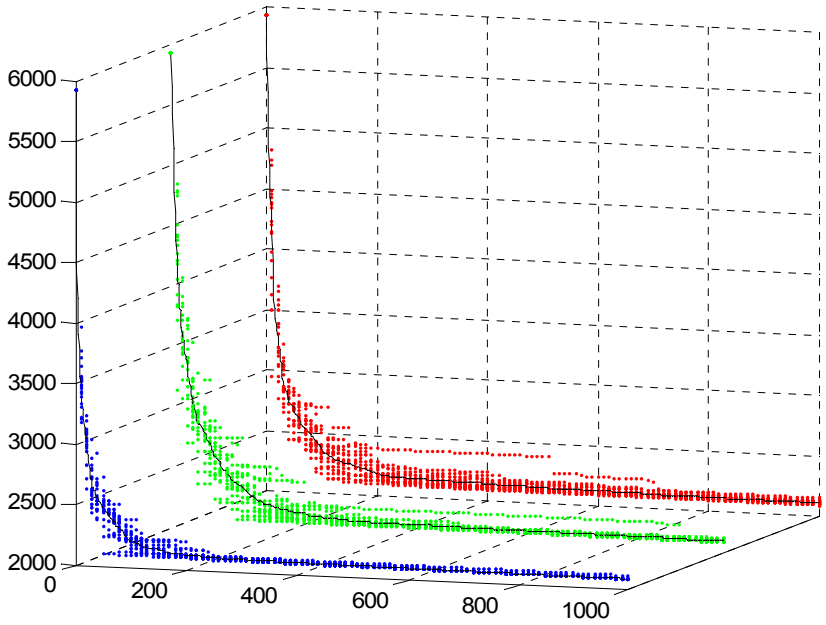


Figura 6.32 Os três melhores operadores de mutação.

Analisando os gráficos apresentados na figura 6.32, verificamos que a dispersão dos valores relativamente à média é menor no operador *arbitrary 20%-job change*. Assim, podemos concluir que este operador, relativamente aos restantes, tem um comportamento mais estável (previsível).

O gráfico mostrado na figura 6.33 apresenta a média do tempo do CPU, em segundos, dos três melhores operadores de mutação obtidos nos testes computacionais. Os operadores *arbitrary 20%-job change* e *arbitrary two-job change* têm um custo computacional muito semelhante (ver figura 6.33) mas, o *arbitrary 20%-job change* tem um custo computacional ligeiramente maior. Por esta razão, não é muito perceptível esta diferença no gráfico da figura 6.33, embora ela exista. Para o operador *arbitrary three-job change*, a melhor performance foi obtida para uma população de 60 cromossomas (ver tabela 6.6) e por essa razão este operador apresenta um custo computacional menor.

Concluímos que o operador *arbitrary 20%-job change* é o que apresenta melhor desempenho ao longo das 1000 gerações (ver figura 6.29 e 6.32) mas, a figura 6.33 mostra que também é o que tem um custo computacional maior.

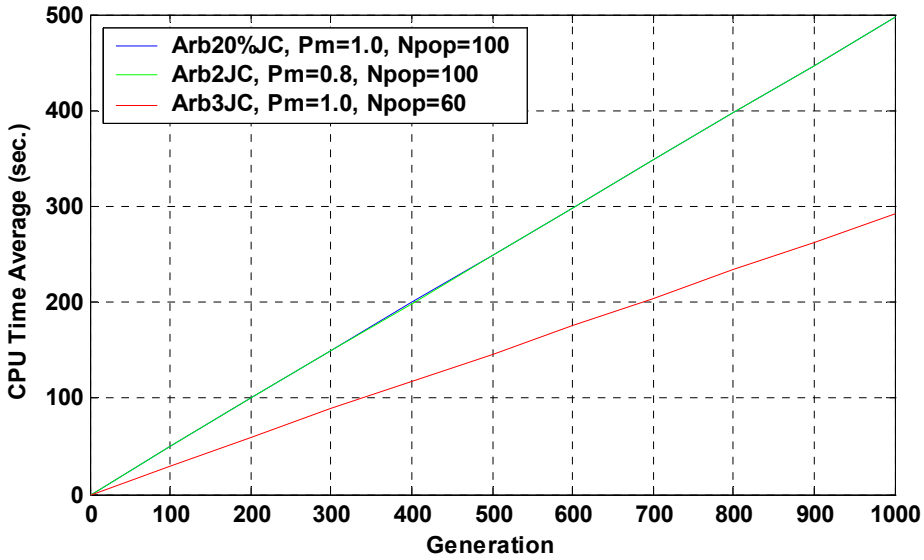
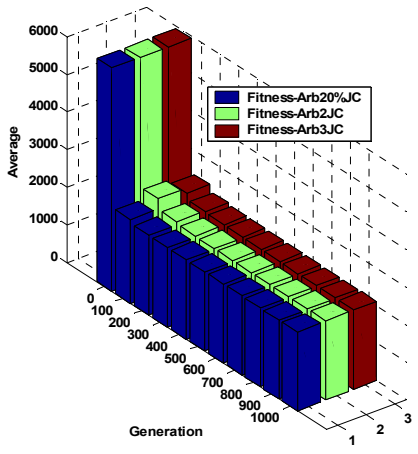
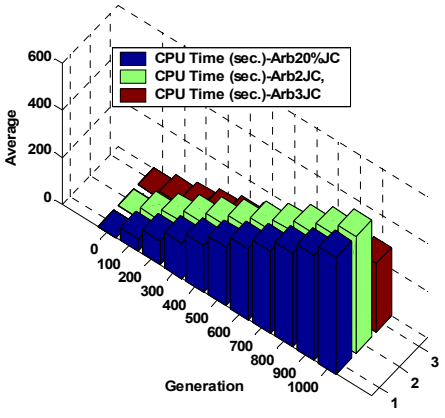


Figura 6.33 Média do tempo do CPU.

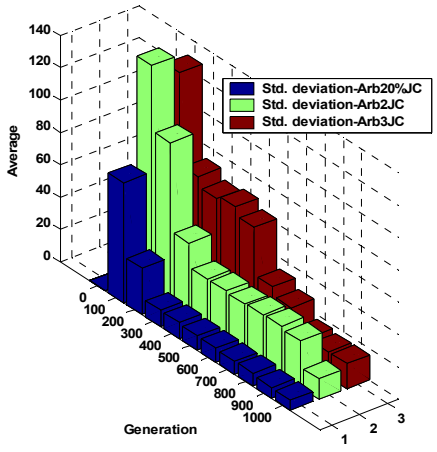
Os três gráficos apresentados na figura 6.34, correspondentes aos três melhores operadores de mutação, apresentam a média do *fitness*, a média do tempo do CPU (em segundos) e a média do desvio padrão obtidas nos testes computacionais realizados. Em todos os gráficos, as barras representam os resultados obtidos nos testes computacionais de 100 em 100 gerações. A figura 6.34(a) mostra que o operador *arbitrary 20%-job change* tem uma evolução muito rápida nas primeiras 100 gerações mas, também é verdade que este necessita de mais tempo de CPU (ver figura 6.34(b)) relativamente aos outros operadores. Por outro lado, a figura 6.34(c) mostra que o operador *arbitrary 20%-job change* é o que apresenta ao longo das 1000 gerações o valor médio de desvio padrão mais baixo. Isto significa que a dispersão dos valores obtidos (nos 20 testes computacionais) relativamente à média do *fitness* é mais pequena neste operador (*arbitrary 20%--job change*) [Ferrolho *et al.*, 2006a] e [Ferrolho *et al.*, 2006b].



a)



b)



c)

Figura 6.34 Os três melhores operadores de mutação:
a) Média do *fitness* b) Média do CPU (seg.) c) Média do desvio padrão

6.5.4 Combinação dos operadores genéticos

Baseados nos resultados obtidos nas tabelas 6.4 e 6.6, aplicámos os melhores operadores genéticos obtidos nos testes computacionais (*two-point crossover: 4 children* com P_c=1,0 e *arbitrary 20%-job change* com P_m=1,0) no HybFlexGA. O objectivo foi avaliar o desempenho resultante da combinação dos dois melhores operadores genéticos. À semelhança dos testes realizados para os operadores de cruzamento e de mutação, foram efectuados 20 testes, utilizando a mesma população inicial de 100 cromossomas (N_{pop}=100) e a mesma condição de paragem (1000 gerações). Destes testes computacionais, obtivemos uma média de performance de 3836. Este resultado, bem como o obtido na figura 6.35 sugerem a existência de uma combinação positiva entre o operador de cruzamento e de mutação, na medida em que há uma melhoria da média da performance e uma melhoria da média do *fitness* ao longo das 1000 gerações [Ferrolho *et al.*, 2006b].

Murata [Murata *et al.*, 1996] demonstrou que com a combinação de outros operadores genéticos, outros operadores de cruzamento e outros operadores de mutação, por vezes obtêm-se combinações negativas.

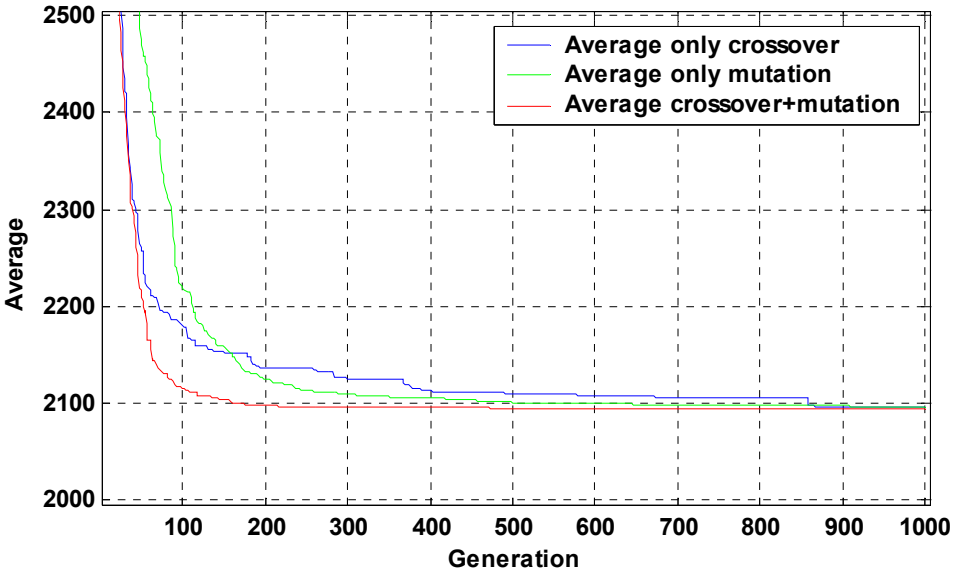


Figura 6.35 Comparação dos melhores operadores genéticos.

6.5.5 Influência da população inicial na qualidade da solução final

Foi realizado um conjunto de testes com o objectivo de verificarmos se a população inicial tem alguma influência na qualidade da solução final. Da análise dos resultados, e como mostra o gráfico da figura 6.36, verificou-se que, de um modo geral a população inicial não influencia significativamente a qualidade da solução final, isto é, partindo de uma solução inicial cujo valor é mais próximo da solução óptima do problema, não significa que se obtenha uma solução melhor do que se partirmos de uma solução inicial pior [Ferrolho *et al.*, 2006a], [Ferrolho e Crisóstomo, 2007b] e [Ferrolho e Crisóstomo, 2007c].

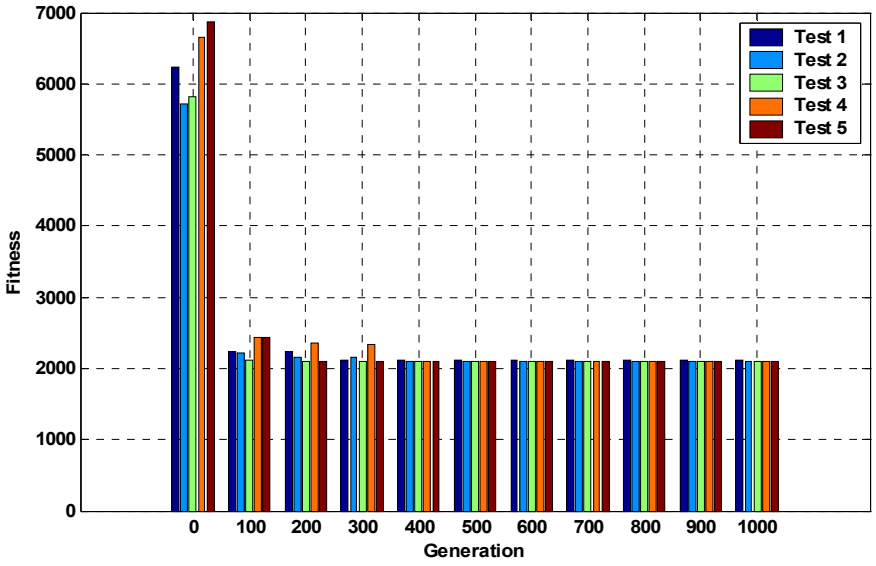


Figura 6.36 Influência da população inicial na qualidade da solução final.

6.5.6 Influência do tamanho da população na qualidade da solução final

O tamanho da população, N_{pop}, é um parâmetro de grande importância em qualquer algoritmo genético, porque condiciona a qualidade da solução obtida e o tempo de processamento. Populações de pequena dimensão poderão tornar-se muito restritivas, em termos de informação genética e condicionarem o algoritmo para uma convergência prematura. Por outro lado, com o aumento do tamanho da população obtém-se uma maior probabilidade de produzir melhores soluções, através duma maior cobertura do espaço de pesquisa, prevenindo a convergência prematura, embora à custa de um maior esforço computacional.

Aproveitámos os testes computacionais realizados para retirarmos algumas conclusões sobre o tamanho da população mais apropriado. Da análise dos seis melhores operadores de cruzamento, obtidos nos testes computacionais realizados, verificámos que, e como referido no parágrafo anterior, o tamanho da população N_{pop} influencia a qualidade da solução final. Da leitura dos gráficos da figura 6.37 observamos que, regra geral, a qualidade da solução final é tanto melhor quanto maior for N_{pop}. Os operadores *position based crossover* e *cycle crossover* (gráficos da figura 6.37(d) e(f)) são excepções à regra, uma vez que apresentam, como seria de esperar, uma solução melhor para N_{pop}=80 e não para N_{pop}=100. Na nossa opinião, e com base na leitura dos gráficos, não é vantajoso trabalhar com tamanhos de população inferiores a 40 cromossomas (N_{pop}<40), porque se obtêm soluções de menor qualidade. Na verdade, é na mudança de 20 para 40 cromossomas que se verifica uma melhoria mais significativa na qualidade das soluções. Também, na nossa opinião, não é vantajoso trabalhar com populações superiores a 100 cromossomas, porque a melhoria da qualidade das soluções não justifica o aumento do custo computacional. Também verificámos que para populações superiores a 60 cromossomas as melhorias das soluções encontradas são pouco significativas, havendo mesmo, por vezes, perca da qualidade das soluções com o aumento do tamanho da população (gráficos (d) e (f) da figura 6.37).

Como conclusão, podemos afirmar que existem vantagens em usarmos tamanhos de populações pertencentes ao intervalo [60, 100], isto é, $N_{pop} \in [60, 100]$ [Ferrolho *et al.*, 2006a], [Ferrolho e Crisóstomo, 2007b].

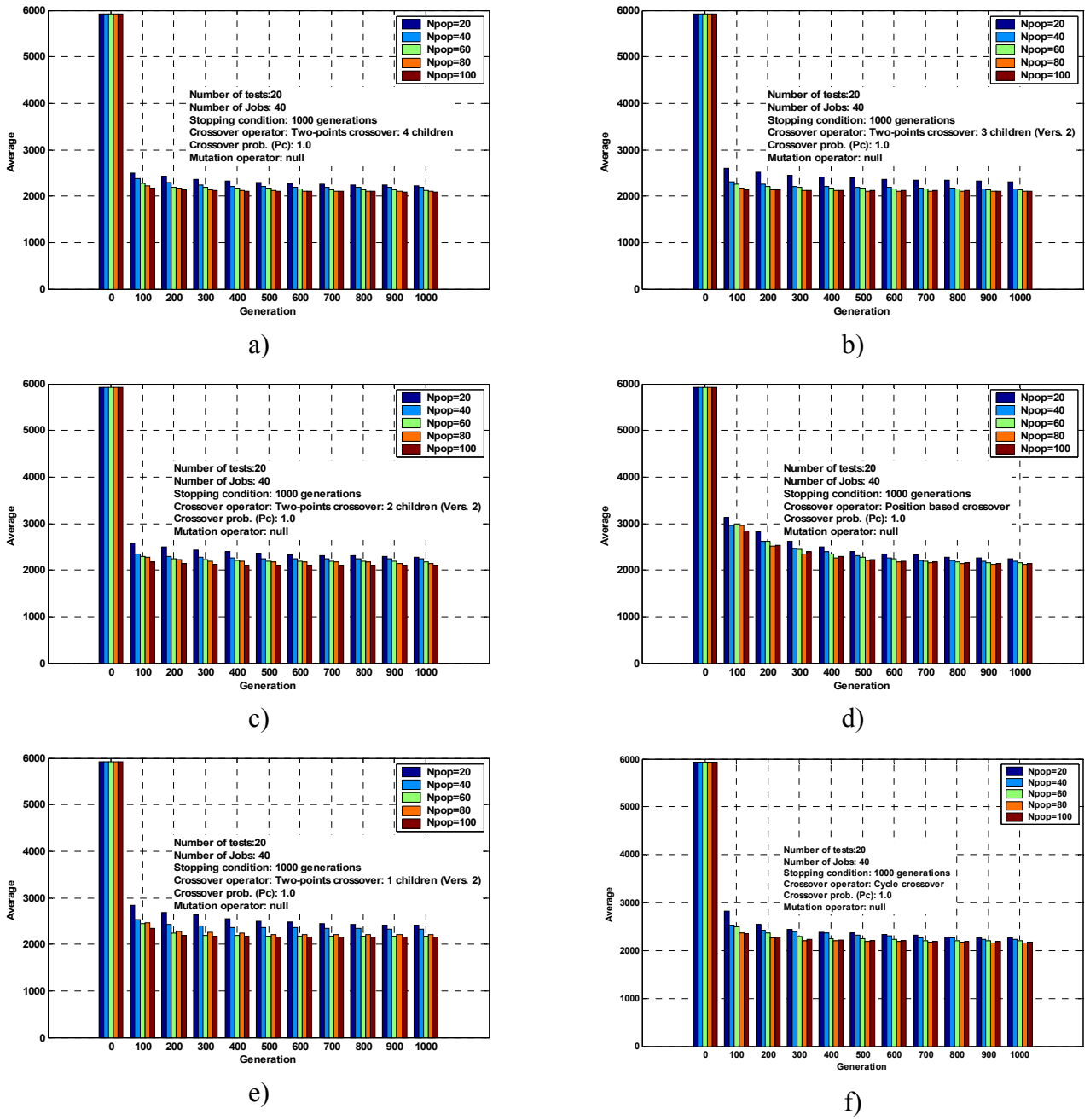


Figura 6.37 Influência do tamanho da população na qualidade da solução final:
a) *Two-point crossover: 4 children* b) *Two-point crossover: 3 children (Ver. II)*
c) *Two-point crossover: 2 children (Ver. II)* d) *Position based crossover*
e) *Two-point crossover: 1 child (Ver. II)* f) *Cycle crossover*

Para os seis melhores operadores de cruzamento apresentados na tabela 6.4, os gráficos apresentados na figura 6.38 mostram o desvio padrão, medida do grau de dispersão dos valores obtidos relativamente à média. O desvio padrão obtido vem reforçar as conclusões a que chegámos sobre a influência do tamanho da população na qualidade da solução final.

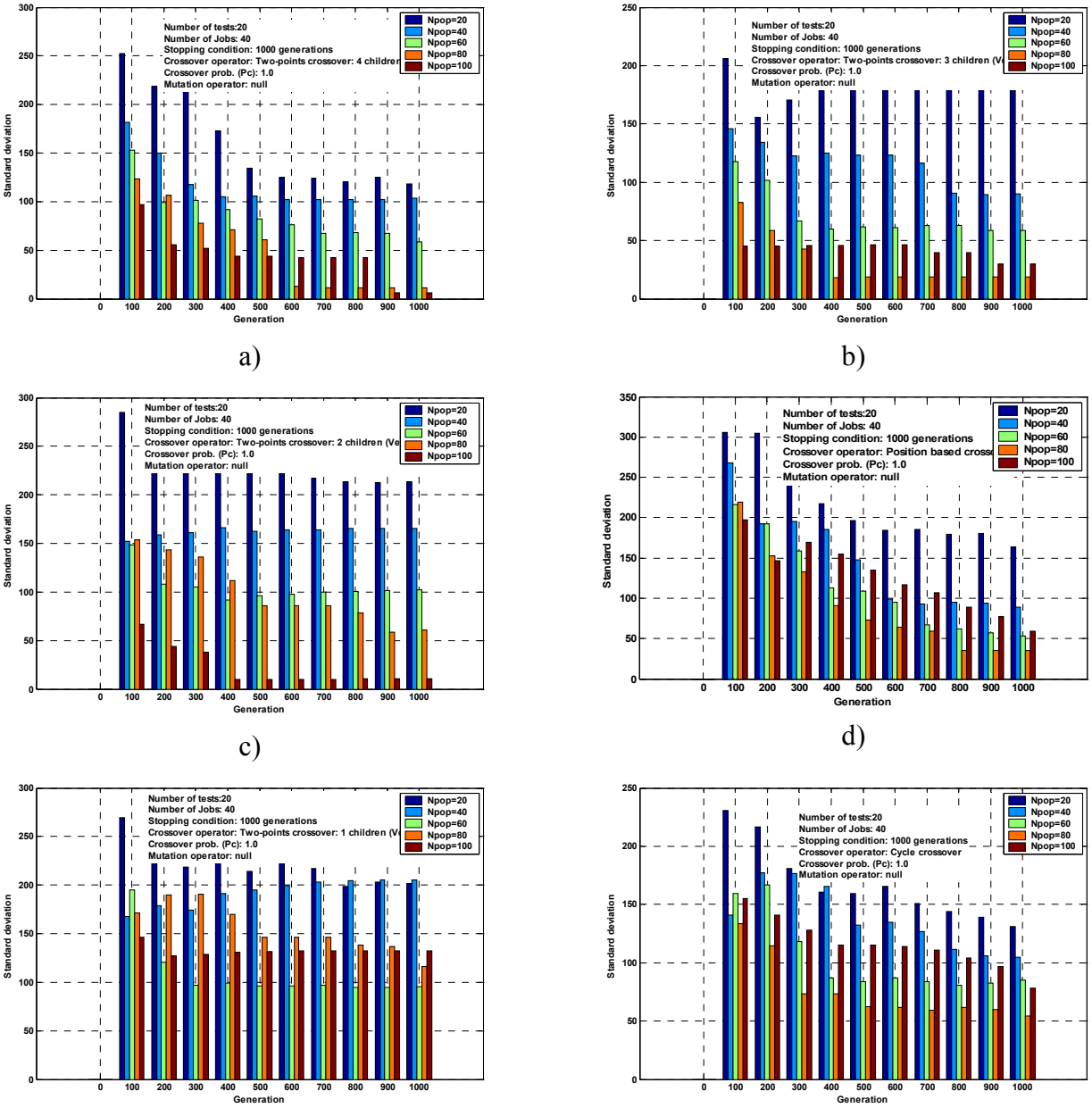


Figura 6.38 Desvio padrão dos seis melhores operadores de cruzamento:
a) *Two-point crossover: 4 children* b) *Two-point crossover: 3 children (Ver. II)*
c) *Two-point crossover: 2 children (Ver. II)* d) *Position based crossover*
e) *Two-point crossover: 1 child (Ver. II)* f) *Cycle crossover*

6.6 Resultados computacionais

Nesta secção apresentamos os resultados computacionais obtidos com instâncias de problemas de 40, 50 e 100 *jobs*. Da OR-Library [http#4], retiramos aleatoriamente algumas instâncias do problema de minimização da soma dos atrasos pesados com uma só máquina (*Single Machine Total Weighted Tardiness Problem* – SMTWT), encontrando-se estas no apêndice A (ver as tabelas A.2.1 até à A.2.7). Foram efectuados 20 testes computacionais para cada uma das

instâncias do problema SMTWT, retiradas da OR-Library [http#4]. Nos testes computacionais realizados no HybFlexGA, usamos os seis melhores operadores de cruzamento obtidos na tabela 6.4 e o melhor operador de mutação obtido na tabela 6.6. Cada instância do problema SMTWT foi examinada de acordo com as seguintes condições:

– Número de testes: 20;
– População inicial (Ψ_t): gerada aleatoriamente;
– Número de *jobs*: 40, 50 e 100;
– Instância usada: da OR-Library [http#4];
– Tamanho da população (N_{pop}): 80 e 100 cromossomas (ver as tabelas 6.4 e 6.6);
– Condição de paragem: 1000 gerações para as instâncias de 40 e 50 *jobs* ou a solução óptima, e 5000 gerações para as instâncias de 100 *jobs* ou a solução óptima;
– Operador de cruzamento: os seis melhores operadores de cruzamento obtidos na tabela 6.4;
– Probabilidade de cruzamento (P_c): 0,8 e 1,0 (ver tabela 6.4);
– Operador de mutação: o melhor operador de mutação obtido na tabela 6.6;
– Probabilidade de mutação (P_m): 1,0 (ver tabela 6.5).

As tabelas 6.8, 6.9 e 6.10 mostram os resultados computacionais obtidos para os problemas SMTWT com 40, 50 e 100 *jobs*. Nestas tabelas temos, para cada instância:

– O número de testes que obtiveram a solução óptima;
– A média do CPU, em segundos, obtida nos testes com solução óptima;
– A média das gerações (arredondada à unidade) obtida nos testes com solução óptima.

Assim, e exemplificando, dos 20 testes computacionais realizados no HybFlexGA à instância 40A (problema SMTWT com 40 *jobs*) com o operador de cruzamento *two-point crossover: 4 children* (TPC4C) e o operador de mutação *arbitrary 20%-job change* (Arb20%JC), obtivemos 16 testes com solução óptima. Nesses 16 testes, a média do CPU foi de 362,4 segundos e a média das gerações foi de 593, como mostra a tabela 6.8 [Ferrolho *et al.*, 2006a].

Como podemos verificar através das tabelas 6.8, 6.9 e 6.10, obtivemos bons resultados com as combinações TPC4C+Arb20%JC, TPC3CV2+Arb20%JC e TPC2CV2+Arb20%JC, para todas as instâncias de 40, 50 e 100 *jobs* analisadas. Mas, como podemos verificar através destas tabelas, os melhores resultados foram obtidos para a combinação TPC4C+Arb20%JC.

Tabela 6.8 Resultados computacionais obtidos para as instâncias de 40 *jobs*

	Instância / **Valor óptimo**	**40A** **6575**	**40B** **1225**	**40C** **6324**	**40D** **6865**	**40E** **990**	**40F** **6955**
TPC4C + A20%JC	Nº de testes c/ solução óptima	16	20	19	20	20	20
	Média do CPU (seg.)	362,4	190,0	319,7	146,1	23,4	214,9
	Média das gerações	593	284	475	239	35	347
TPC3CV2 + A20%JC	Nº de testes c/ solução óptima	13	15	15	20	20	19
	Média do CPU (seg.)	382,9	231,3	260,5	250,2	27,7	261,3
	Média das gerações	725	402	448	474	48	489
TPC2CV2 + A20%JC	Nº de testes c/ solução óptima	8	16	15	17	20	16
	Média do CPU (seg.)	369,1	216,8	305,8	250,7	27,9	292,5
	Média das gerações	380	449	627	563	58	651
PBX + A20%JC	Nº de testes c/ solução óptima	0	8	1	0	20	0
	Média do CPU (seg.)	----	236,1	312,0	----	52,6	----
	Média das gerações	----	747	976	----	168	----
TPC1CV2 + A20%JC	Nº de testes c/ solução óptima	0	3	2	0	20	2
	Média do CPU (seg.)	----	230,0	363,5	----	64,2	331,5
	Média das gerações	----	609	956	----	174	945
CX + A20%JC	Nº de testes c/ solução óptima	0	4	4	4	20	1
	Média do CPU (seg.)	----	165,3	270,5	235,0	34,1	189,0
	Média das gerações	----	551	892	848	115	674

Tabela 6.9 Resultados computacionais obtidos para as instâncias de 50 *jobs*

	Instância / **Valor óptimo**	**50A** **2134**	**50B** **22**	**50C** **2583**	**50D** **2691**	**50E** **4**	**50F** **4674**
TPC4C + A20%JC	Nº de testes c/ solução óptima	20	20	15	20	20	18
	Média do CPU (seg.)	88,3	45,5	214,1	42,7	25,3	573,6
	Média das gerações	107	54	256	56	31	710
TPC3CV2 + A20%JC	Nº de testes c/ solução óptima	18	20	12	20	20	10
	Média do CPU (seg.)	112,3	50,4	207,9	61,2	27,9	489,6
	Média das gerações	158	70	290	94	39	703
TPC2CV2 + A20%JC	Nº de testes c/ solução óptima	17	20	12	20	20	5
	Média do CPU (seg.)	146,2	92,0	157,8	87,3	32,0	467,6
	Média das gerações	246	152	263	160	53	794
PBX + A20%JC	Nº de testes c/ solução óptima	18	20	10	20	20	0
	Média do CPU (seg.)	144,6	86,3	151,6	126,2	46,8	----
	Média das gerações	371	218	385	352	118	----
TPC1CV2 + A20%JC	Nº de testes c/ solução óptima	13	20	5	19	20	0
	Média do CPU (seg.)	224,7	118,2	180,0	142,5	59,3	----
	Média das gerações	487	252	385	335	127	----
CX + A20%JC	Nº de testes c/ solução óptima	17	20	11	20	20	0
	Média do CPU (seg.)	107,8	51,0	140,0	79,4	33,2	----
	Média das gerações	292	136	375	233	88	----

Verificamos que, quando usamos a combinação TPC4C+Arb20%JC, o HybFlexGA é muito eficiente. Por exemplo, nas seis instâncias usadas nos testes com 40 *jobs* (ver tabela 6.8) o HybFlexGA encontrou 20 testes com solução óptima em 4 instâncias (40B, 40D, 40E e 40F), 19 testes com solução óptima numa instância (40C) e 16 testes com solução óptima numa instância

(40A). Esta combinação de operadores genéticos obteve também bons resultados nos problemas SMTWT com 50 e 100 *jobs* (ver tabelas 6.9 e 6.10). Os resultados obtidos e apresentados nas tabelas 6.8, 6.9 e 6.10 mostram que o HybFlexGA tem uma boa performance e eficiência com a combinação TPC4C+Arb20%JC.

Tabela 6.10 Resultados computacionais obtidos para as instâncias de 100 *jobs*

	Instância / **Valor óptimo**	**100A** **5988**	**100B** **8**	**100C** **4267**	**100D** **5011**	**100E** **5283**	**100F** **50**
TPC4C + A20%JC	Nº de testes c/ solução óptima	16	20	19	20	20	20
	Média do CPU (seg.)	2405,1	523,9	3213,5	2428,1	2958,8	1113,9
	Média das gerações	1611	323	1971	1483	1808	692
TPC3CV2 + A20%JC	Nº de testes c/ solução óptima	15	20	15	10	20	20
	Média do CPU (seg.)	3851,0	1012,4	4453,7	2841,3	3759,9	1891,2
	Média das gerações	3042	727	3181	2021	2677	1376
TPC2CV2 + A20%JC	Nº de testes c/ solução óptima	9	20	5	5	18	17
	Média do CPU (seg.)	3921,3	1339,8	4036,2	4406,6	3937,6	2890,6
	Média das gerações	3685	1142	3423	3725	3335	2507
PBX + A20%JC	Nº de testes c/ solução óptima	1	20	3	0	2	13
	Média do CPU (seg.)	2067,0	1413,6	3237,7	----	3548,5	2174,2
	Média das gerações	2957	1828	4182	----	4569	2843
TPC1CV2 + A20%JC	Nº de testes c/ solução óptima	1	19	2	0	3	6
	Média do CPU (seg.)	4104,0	2190,7	3847,0	----	4070,3	3185,5
	Média das gerações	4948	2405	4192	----	4418	3522
CX + A20%JC	Nº de testes c/ solução óptima	3	20	5	3	11	19
	Média do CPU (seg.)	2594,0	736,4	3224,2	2463,3	2415,0	1770,7
	Média das gerações	3912	1011	4390	3349	3282	2440

Nas subsecções 6.4.2 e 6.4.3 ficou demonstrado que o operador de cruzamento *two-point crossover: 4 children* e o operador de mutação *arbitrary 20%-job change* necessitam de mais tempo de CPU (para o mesmo número de gerações) do que os outros operadores genéticos. Mas, também ficou demonstrado que estes operadores necessitam de um menor número de gerações para encontrarem boas soluções, havendo mesmo uma grande probabilidade de encontrarem a solução óptima. Os resultados computacionais obtidos nas tabelas 6.8, 6.9 e 6.10 mostram que o HybFlexGA com a combinação TPC4C+Arb20%JC requer menos tempo de CPU comparativamente às outras combinações. Por exemplo, na instância 50B, o HybFlexGA encontrou sempre as 20 soluções óptimas em todas as combinações (TPC4C+Arb20%JC, TPC3CV2+Arb20%JC, TPC2CV2+Arb20%JC, etc.). Para esta instância, como podemos verificar na tabela 6.9, a combinação que necessitou de menos tempo de CPU foi a TPC4C+Arb20%JC (valor médio de 45,5 segundos). Todas as restantes combinações necessitaram de mais tempo de CPU.

6.7 Sistema de sequenciamento dinâmico

Nesta secção é descrita e apresentada a arquitectura desenvolvida para a resolução de problemas de sequenciamento dinâmicos, baseados em algoritmos genéticos. Muitas vezes, os planos de sequenciamento obtidos são válidos durante um curto período de tempo devido ao aparecimento de acontecimentos inesperados que alteram o estado do sistema de produção. Em situações complexas, típicas da realidade, podemos afirmar que o sequenciamento é um processo contínuo de sucessivos resequenciamentos.

Em [Rabelo e Camarinha-Matos, 1994] e [Rabelo, 1997] os autores consideram que:

> "S*equenciamento dinâmico é a situação na qual se executa um plano flexível, ou seja, o plano inicial é dinamicamente ajustado enquanto os eventos (normalmente inesperados, tais como: avarias das máquinas, alterações da prioridade das ordens, atraso em algumas operações) vão acontecendo no ambiente de produção, e tal plano de sequenciamento permanece realista (reflectindo o que foi planeado) e exequível (coerente de acordo com os objectivos e restrições actuais) durante o seu processamento*".

Graves [Graves, 1981], num estudo realizado em 1981, faz a seguinte observação, aparentemente ainda muito actual:

> "*A geração de um plano pode ser fácil, o que é difícil é a sua permanente revisão, requerida pelo dinamismo do sistema*".

Um sistema real de produção, como por exemplo, a Célula Flexível de Fabrico (CFF) apresentada no capítulo quatro, está sujeito à alteração de vários factores internos e externos. Assim, podem ocorrer acontecimentos aleatórios e não previsíveis que provocam uma alteração do estado do sistema. Como exemplo de tais acontecimentos, podemos citar o lançamento de novas ordens de fabrico ou o cancelamento de algumas outras; a alteração de prioridade de ordens já lançadas; atrasos no processamento das operações, e ainda, a alteração da disponibilidade de equipamento devido a avarias. Tais acontecimentos invalidam a eficácia de algumas abordagens estáticas aos problemas de sequenciamento em sistemas de fabrico reais, uma vez que o plano de sequenciamento inicialmente obtido deixa de ser válido quando o cenário inicial previsto é modificado. Assim, sempre que ocorre uma perturbação no sistema é necessário reajustar o sequenciamento, isto é, fazer um resequenciamento (*rescheduling*). Em [Smith, 1994], o resequenciamento é definido como o processo de rever ou reestruturar um plano de sequenciamento que se tenha tornado desactualizado devido à ocorrência de acontecimentos

inesperados no sistema. Assim, o resequenciamento é o processo de adaptar o plano de sequenciamento ao novo estado do sistema, preservando tanto quanto possível o plano anterior e reagindo o mais rapidamente possível aos eventos que tenham ocorrido. O resequenciamento é também conhecido como sequenciamento reactivo, sequenciamento em tempo real, ou sequenciamento dinâmico. Mas, ao longo desta tese optámos por utilizar o termo resequenciamento por, na nossa opinião, ser aquele que se aproxima mais do termo inglês – *rescheduling*.

A figura 6.39 apresenta a arquitectura do sistema de sequenciamento dinâmico implementada no HybFlexGA. Esta arquitectura é composta por quatro módulos: interface com o utilizador, pré--processamento, sequenciamento com AG e adaptação dinâmica. Na verdade, esta arquitectura possui mais um módulo (adaptação dinâmica) relativamente à arquitectura apresentada na figura 6.20 da secção 6.3.

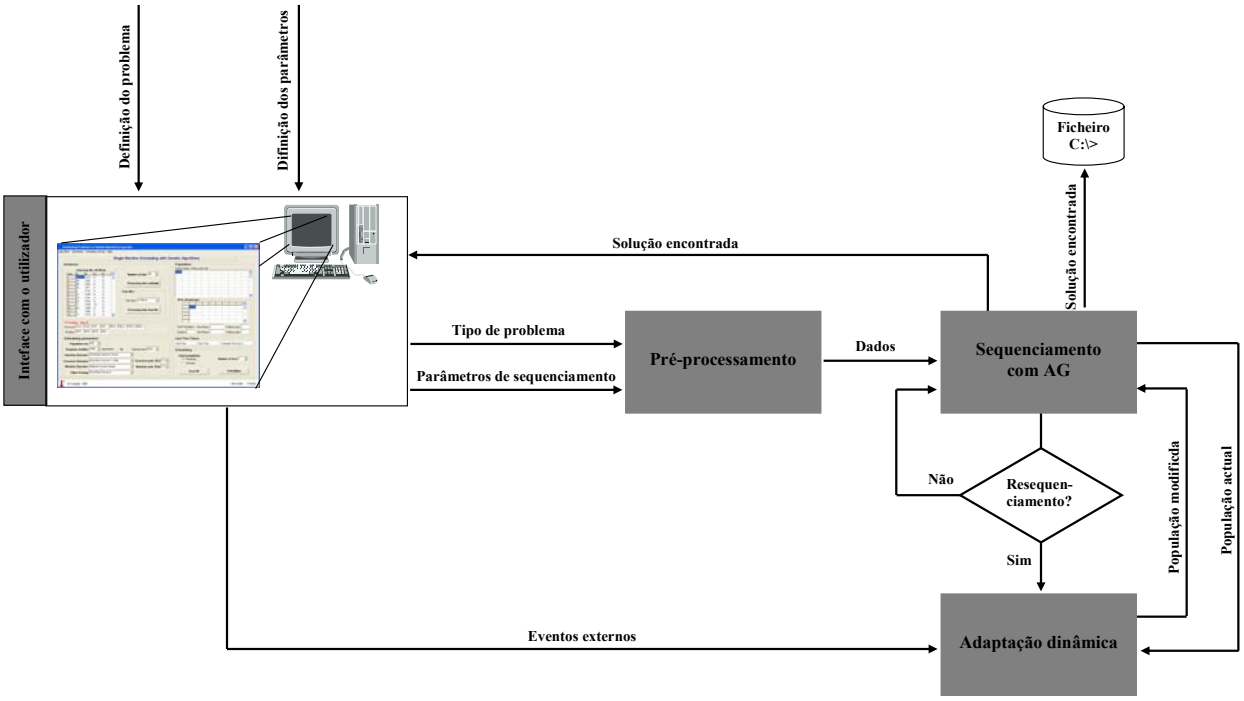


Figura 6.39 Arquitectura do sistema de sequenciamento dinâmico.

Na secção 6.3 foram apresentados os módulos interface com o utilizador, pré-processamento e sequenciamento com AG. Estes módulos também foram utilizados na arquitectura do sistema de sequenciamento dinâmico (ver figura 6.39). Na subsecção 6.7.2 será apresentado o módulo de adaptação dinâmica usado nesta arquitectura.

6.7.1 Estratégias de resequenciamento

Num ambiente dinâmico, caracterizado por ocorrerem normalmente eventos aleatórios, existe uma grande variação nas condições e requisitos de processamento ao longo do tempo. Após a ocorrência de novos eventos no sistema, é possível usar as seguintes estratégias:

- Reiniciar o módulo de sequenciamento com AG, com a nova população inicial de cromossomas actualizada;
- Continuar o módulo de sequenciamento com AG, depois de actualizada a população actual de cromossomas, no módulo de adaptação dinâmica (ver figura 6.39).

Se o processo de sequenciamento ainda não tiver sido iniciado, é evidente que a maneira mais simples de abordar o problema de resequenciamento é recomeçar o módulo de sequenciamento com AG a partir dos novos dados do problema, isto é, tendo em conta as alterações ocorridas em relação ao problema inicial de sequenciamento. Assim, tendo em conta as alterações ocorridas, é gerada uma nova população inicial de cromossomas. Esta forma de resequenciamento não é a mais eficiente em problemas de sequenciamento reais constantemente sujeitos a diferentes mudanças ou perturbações ao longo do tempo, uma vez que obriga a sucessivos recomeços do módulo de sequenciamento com AG e, consequentemente, perda de informação da população actual de cromossomas.

Após o problema de sequenciamento ter iniciado, devemos utilizar uma estratégia que reutilize toda a informação existente e processada até esse momento. Em problemas de sequenciamento reais de significativa dimensão, o resequenciamento a partir do início deve ser evitado, porque a frequência com que surge a necessidade de resequenciamento é grande e, por outro lado, os tempos de processamento envolvidos são elevados. Assim, deve ser tida em conta, a população actual de cromossomas gerada pelo módulo de sequenciamento com AG. Esta população será actualizada no módulo de adaptação dinâmica, com base nos novos eventos ocorridos.

6.7.2 Módulo de adaptação dinâmica

O módulo de adaptação dinâmica tem como objectivo assegurar a viabilidade do sequenciamento dinâmico. Sempre que um *job* é introduzido ou cancelado no sistema, isto é, sempre que ocorre um novo evento, a estrutura dos cromossomas deverá ser modificada, aumentando ou diminuindo de tamanho, de forma a reflectirem o estado actual do problema.

Se um *job* é introduzido no sistema, é necessário introduzi-lo, também, na população de cromossomas actual. A introdução dos novos *jobs* nos vários cromossomas da população pode ser feita das seguintes formas:

– Aleatoriamente – consiste em seleccionar aleatoriamente uma posição e inserir o novo *job* nessa posição em todos os cromossomas da população. Caso haja mais do que um *job* a inserir, devem ser seleccionadas aleatoriamente posições diferentes para cada *job*;

– *First In First Out* (FIFO) – consiste em inserir os novos *jobs*, pela ordem que vão chegando, no fim da sequência dos cromossomas.

Quando um *job* é cancelado é necessário eliminá-lo em todos os cromossomas da população. Este processo de cancelamento de *jobs* nos cromossomas dá origem ao aparecimento de lacunas e, assim, todos os *jobs* posicionados à direita das lacunas irão sofrer um deslocamento para a esquerda. Este deslocamento tem como objectivo a ocupação das lacunas deixadas pelos *jobs* eliminados e consequentemente o tamanho dos cromossomas é reduzido.

A figura 6.40 apresenta a interface com o utilizador do HybFlexGA. Nesta interface, o botão *ReScheduling* permite executar o resequenciamento, estando-lhe associado o módulo de adaptação dinâmica apresentado na figura 6.39.

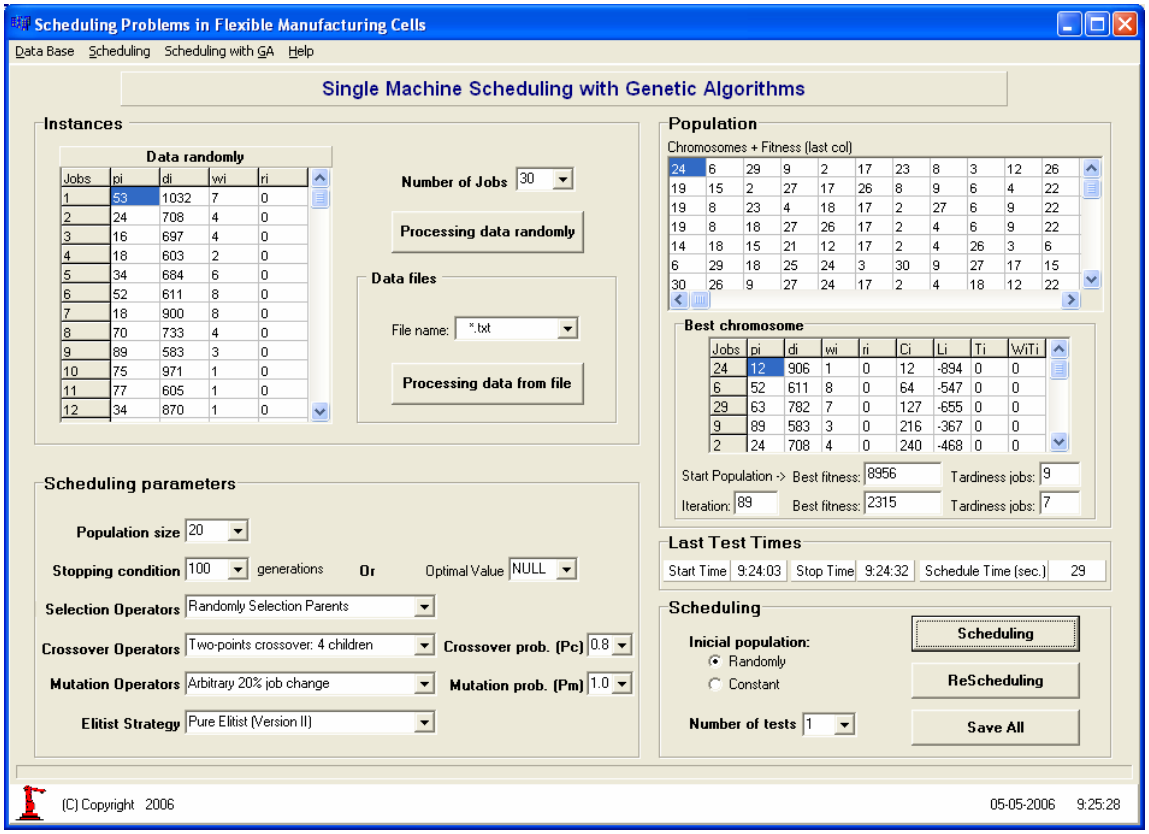


Figura 6.40 Interface desenvolvida para o sequenciamento dinâmico.

6.8 Resumo do Capítulo

Neste capítulo apresentamos um novo conceito de operadores genéticos para problemas de sequenciamento. Desenvolvemos o software HybFlexGA para examinarmos o desempenho dos vários operadores de cruzamento e de mutação, utilizando testes computacionais em problemas de sequenciamento. Os testes computacionais realizados demonstraram que o operador de cruzamento *two-point crossover: 4 children* e o operador de mutação *arbitrary 20%-job change* são os que apresentam melhor desempenho. Também demonstramos que a combinação destes dois operadores genéticos é positiva, uma vez que tal combinação resulta numa melhoria do desempenho.

O HybFlexGA obteve bons resultados computacionais nas instâncias do problema SMTWT com 40, 50 e 100 *jobs* (ver tabelas 6.8, 6.9 e 6.10). Como ficou demonstrado, o HybFlexGA é muito eficiente com a combinação TPC4C+Arb20%JC. Com esta combinação de operadores genéticos, o HybFlexGA encontrou, sempre, mais soluções óptimas (em todas as instâncias do problema SMTWT usadas neste capítulo) do que com outras combinações, nomeadamente, TPC3CV2+Arb20%JC, TPC2CV2+Arb20%JC, entre outras. Quando o operador de cruzamento *two-point crossover: 4 children* é combinado com o operador de mutação *arbitrary 20%-job change* no HybFlexGA, o algoritmo genético requer menos gerações e menos tempo de CPU para encontrar soluções óptimas.

O módulo de adaptação dinâmica implementado no HybFlexGA permite a resolução de situações de sequenciamento dinâmico (resequenciamento). Assim, sempre que ocorrem novos eventos (introdução ou cancelamento de *jobs*) ou perturbações (alteração dos tempos de processamento, datas de entrega, etc.) o HybFlexGA tem a capacidade de reagir e de se adaptar às mudanças ocorridas durante o sequenciamento.

No capítulo seguinte iremos abordar os problemas de sequenciamento com mais de uma máquina.

Capítulo 7

Algoritmos genéticos aplicados aos problemas de sequenciamento com duas ou mais máquinas

"O que quer que seja que tu faças será insignificante,
mas é importante que o faças."
"Mahatma" Gandhi (1869-1948),
líder político e espiritual Indiano.

7.1 Introdução

Como foi referido no capítulo cinco, os problemas de sequenciamento podem ser enquadrados em duas categorias [French, 1982], [Gaither, 1989] e [Ranky, 1990]:

- de acordo com o que será produzido:
  - Determinísticos: quando os tempos de processamento e todos os demais parâmetros são conhecidos e fixos;
  - Não-determinísticos (ou estocásticos): quando os tempos de processamento e os demais parâmetros não são (todos) conhecidos.
- de acordo com o ambiente de sequenciamento:
  - Estáticos: quando o número de *jobs* e os respectivos tempos em que eles estarão disponíveis para o início do processamento (*ready-times*) são conhecidos e fixos;
  - Dinâmicos: quando os *jobs* chegam aleatoriamente num certo período de tempo, ou seja, o número de *jobs* é desconhecido.

De uma maneira geral, um ambiente estocástico/dinâmico está relacionado com indústrias em que o tipo de produção é flexível e por encomenda. Já o ambiente determinístico/estático está relacionado com a indústria de produção em série.

Neste capítulo, serão aplicados os algoritmos genéticos a problemas de sequenciamento, com duas ou mais máquinas, do tipo *Job-Shop*. Na verdade, o *Job-Shop* destaca-se dos restantes problemas de sequenciamento, por ser aquele que mais se aproxima da realidade industrial e, por

isso, tem sido alvo de uma quantidade significativa de trabalhos de investigação levados a cabo até aos nossos dias. Assim, optamos por este tipo de problemas de sequenciamento pelo facto destes serem os mais comuns nos sistemas de fabrico real, mas também por serem os de mais difícil resolução.

O problema *Job-Shop* clássico consiste num conjunto de máquinas diferentes capazes de processarem as operações dos *jobs*. Cada *job* é constituído por um conjunto de operações, sendo cada operação processada numa máquina diferente. Os problemas do tipo *Job-Shop* mais frequentes na literatura, muitas vezes designados como problemas académicos, consideram que cada *job* possui *m* operações, uma em cada máquina, ou seja, cada *job* passa por cada uma das máquinas uma única vez, resultando assim em *n* x *m* operações [French, 1982]. Sendo fixa a sequência de processamento das operações de cada *job* nas máquinas, o problema *Job-Shop* clássico consiste em encontrar a sequência mais adequada para o processamento dos *jobs* nas máquinas, minimizando o tempo total de fabrico dos *jobs*, i.e., o *makespan*. Este é um problema de resolução muito complexa, sendo conhecido na literatura como NP difícil (*NP-hard*) [Lawler *et al.*, 1993], [Jain *et al.*, 1999].

Este capítulo está estruturado da seguinte forma: a secção 7.2 expõe os problemas de sequenciamento *Job-Shop*; na secção 7.3 encontra-se um resumo dos trabalhos mais relevantes encontrados na literatura; a secção 7.4 apresenta a notação utilizada no sequenciamento *Job-Shop*; a secção 7.5 aborda o modelo de sequenciamento proposto para a resolução de problemas de sequenciamento *Job-Shop*; a secção 7.6 apresenta três exemplos de sequenciamento *Job-Shop* com o objectivo de ilustrar e explicar o funcionamento do algoritmo de sequenciamento proposto; a secção 7.7 mostra e compara um conjunto de testes e resultados computacionais obtidos pelo modelo de sequenciamento proposto e por vários algoritmos desenvolvidos por outros autores e, por último, a secção 7.8 apresenta as conclusões do trabalho desenvolvido ao longo deste capítulo.

7.2 O Problema de Sequenciamento *Job-Shop*

O problema de sequenciamento *Job-Shop* é um problema combinatório de elevada complexidade, daí que o tempo requerido para encontrar a solução óptima aumenta exponencialmente com o tamanho do problema. Um problema de sequenciamento de *n jobs* em *m* máquinas pode envolver $(n!)^m$ sequências possíveis, como já foi referido anteriormente.

Assim, por exemplo, o número de sequências para um problema de 20 *jobs* e 10 máquinas pode ser $(20!)^{10}$, i.e., $7{,}27 \times 10^{183}$. Muitas destas sequências não são certamente soluções do problema por violarem as suas restrições. Mesmo assim, o número de soluções pode ser muito elevado, tornando-se inviável a pesquisa combinatória para obter a solução óptima. Devido à complexidade exponencial, o problema de sequenciamento *Job-Shop* é incluído numa vasta classe de problemas numéricos intratáveis, conhecida como problemas NP difíceis, como se referiu acima. Esta classe de problemas inclui problemas para os quais não são conhecidos algoritmos eficientes de resolução devido à sua complexidade exponencial. Sendo assim, a resolução destes passa pela utilização de algoritmos, de natureza heurística, que permitem obter boas soluções.

Os problemas de sequenciamento *Job-Shop* são caracterizados da seguinte forma:

- Um conjunto de *n jobs* $J = \{J_i \mid i \in N(n)\}$ multi-operação estão disponíveis para processamento, requerendo cada *job m* operações e cada operação uma máquina diferente. *N(n)* é um subconjunto do conjunto dos números naturais de 1 até *n*;
- Está permanentemente disponível um conjunto de *m* máquinas diferentes $M = \{M_j \mid j \in N(m)\}$, sendo *N(m)* um subconjunto do conjunto dos números naturais de 1 até *m*;
- Os tempos de preparação das operações estão incluídos nos tempos de processamento;
- As operações não podem ser interrompidas, ou seja, uma vez iniciado o seu processamento, estas são processadas até ao fim sem interrupções;
- Cada máquina só pode processar um *job* de cada vez;
- Cada *job* só pode ser processado numa máquina de cada vez;
- Não há restrições de precedência entre operações de *jobs* diferentes;
- Todos os atributos que descrevem os *jobs* são conhecidos à priori (por exemplo: o tempos de processamento e as datas de entrega);
- O ambiente de fabrico é estático, ou seja, não são consideradas chegadas de novos *jobs* nem a avaria das máquinas.

Ao problema de sequenciamento *Job-Shop* estão associadas duas dificuldades. Uma é a sua complexidade exponencial devido à explosão combinatória, como já se referiu anteriormente. Assim, um problema de *n jobs* e *m* máquinas pode envolver no limite $(n!)^m$ soluções possíveis. Para este problema ainda não se conhecem métodos eficientes para a sua resolução. Assim, mesmo para um pequeno número de *jobs* e máquinas, um número proibitivo de situações teria

teoricamente que ser verificado caso se pretendesse obter a solução óptima do problema. A outra dificuldade, é a diversidade de restrições a que o problema está sujeito, como por exemplo, disponibilidade de recursos, datas de entrega dos *jobs*, restrições de precedência entre *jobs*, entre outras.

Uma extensão do problema clássico de sequenciamento *Job-Shop* é o problema *flexible Job--Shop* (*Job-Shop* flexível). Neste tipo de problema é permitido o processamento de uma operação em qualquer uma das máquinas de um determinado conjunto, com o objectivo de minimizar o tempo máximo de conclusão, ou seja o *makespan*. O problema consiste em afectar cada operação a uma máquina (problema de afectação) e ordenar as operações nas máquinas (problema de sequenciamento) de forma a minimizar o tempo máximo de conclusão de todos os *jobs* [Mastrolilli, 1998]. Este problema é mais complexo do que o *Job-Shop* clássico porque para além do sequenciamento das operações existe também a necessidade de resolver o problema de afectação [Jansen *et al.*, 2000]. Este problema é também NP difícil [Garey *et al.*, 1976].

As primeiras abordagens ao problema de sequenciamento *Job-Shop* remontam a 1956 com o algoritmo de Jackson [Jackson, 1956], desenvolvido para resolver problemas do tipo $n/2/G/F_{\max}$. De então para cá, e uma vez que o problema tem suscitado extensa investigação, vários métodos têm sido desenvolvidos. O método heurístico *Shifting Bottleneck*, desenvolvido inicialmente por Adams [Adams *et al.*, 1988] e mais recentemente melhorado por Dauzére-Pérés [Dauzére-Pérés *et al.*, 1993], é um dos métodos marcantes na resolução do problema *Job-Shop*.

7.3 Revisão da Literatura

Nesta secção é apresentado um resumo de alguns trabalhos encontrados na literatura. Apesar da grande quantidade de algoritmos de sequenciamento desenvolvidos até ao momento, existem dificuldades em aplicá-los à realidade industrial, uma vez que estes tratam, na sua maioria, problemas de sequenciamento estáticos e determinísticos. Devido ao dinamismo que caracteriza os sistemas de fabrico reais, as soluções encontradas ficam rapidamente desactualizadas.

Nos últimos anos, um elevado número de heurísticas foram desenvolvidas e aplicadas à resolução de problemas de sequenciamento *Job-Shop*. Como exemplos referimos a pesquisa tabu [Nowicki e Smutnicki, 1996], [Amico *et al.*, 1993], *simulated annealing* [Kolonko, 1999], [Van *et al.*, 1992], colónia de formigas [Colorni *et al.*, 1994], redes neuronais [Foo e Takefuji, 1988],

[Yang, 2006], *branch-and-bound* [French, 1982], *shifting bottleneck heuristic* [Pinedo, 2002], [Dileep, 1996], algoritmos genéticos [Wu *et al.*, 2004], [Mikkel, 2003], [Lui e Xi, 2006], entre outros. Pesquisa tabu (t*abu search*) utiliza um mecanismo de "memória" que força o algoritmo a explorar novas áreas do espaço de pesquisa. As soluções examinadas recentemente são memorizadas (soluções tabu – proibidas) e não são consideradas na escolha da solução seguinte a pesquisar. *Simulated annealing* é uma estratégia de optimização inspirada num modelo de evolução termodinâmica, modelando o processo de aquecimento e arrefecimento lento com vista a encontrar um estado de energia mínimo. Colónia de formigas (a*nt system*) foi inspirada em colónias de formigas reais que depositam uma substância química (feromona) no chão. Quanto maior for a quantidade de feromona existente num determinado caminho maior é a probabilidade da formiga o escolher. Redes neuronais (*neural network*) tentam implementar num computador a capacidade de processamento de dados do cérebro. *Branch-and-bound* pode obter bons resultados na resolução de problemas de sequenciamento *Job-Shop*, mas é computacionalmente exigente. Uma das heurísticas com mais sucesso na minimização do *makespan* em problemas de sequenciamento *Job-Shop* é *shifting bottleneck heuristic*. Nesta heurística, cada máquina por sequenciar é considerada como uma máquina separada, e a máquina com maior atraso é considerada máquina estranguladora (*bottleneck machine*), sendo sequenciada em primeiro. Os algoritmos genéticos são utilizados com sucesso na resolução de problemas de sequenciamento, como mostra a grande quantidade de artigos publicados nos últimos anos.

O software LEKIN [http#5] é um protótipo académico desenvolvido na *Stern School of Business* da *New York University*, sob a orientação e supervisão de Michael Pinedo [Pinedo e Chao, 1998] e [Pinedo, 2002]. Este software de sequenciamento permite a resolução de diferentes classes de problemas de sequenciamento, nomeadamente problemas de sequenciamento com uma só máquina, *Flow-Shop*, *Job-Shop* e *Flexible Job-Shop*. No LEKIN estão implementados alguns dos métodos de sequenciamento mais conhecidos, como por exemplo, regras de prioridade, o método *shifting bottleneck*, entre outros. De referir que o LEKIN contém alguns problemas de teste com a finalidade de permitir averiguar a eficiência e eficácia dos métodos implementados, através da análise de algumas medidas de desempenho. Em [Pinedo e Chao, 1998], [Pinedo, 1995] e [Pinedo, 2002] são ainda referidos outros sistemas de sequenciamento académicos e comerciais.

Muitos outros sistemas ou protótipos académicos foram concebidos, ou estão ainda em desenvolvimento, para ambientes de fabrico do tipo *Job-Shop*, dos quais destacamos: MADEMA

[Chryssolouris, 1987], GPSS [http#6], O-Plan2 [Drabble *et al.*, 1992], OPTIMUM-AIV [Arentoft *et al.*, 1991], PROTOS [Sauer, 1991], CORTES [Fox e Sycara, 1990], SPIKE [Johnson e Miller, 1994], CORA [http#7], entre tantos outros.

Dimopoulos e Zalzala em [Dimopoulos e Zalzala, 2000] analisam os recentes desenvolvimentos no campo da computação evolucionária para problemas de optimização na área da produção. Neste trabalho, os autores fazem um resumo dos problemas tratados, desde os problemas tradicionais do tipo *Job-Shop* e *Flow-Shop*, até aos problemas de balanceamento de linhas. Os autores apresentam os estudos computacionais realizados e respectivos resultados obtidos. Neste trabalho, os autores também referem alguns trabalhos realizados na tentativa de resolverem o problema de sequenciamento dinâmico.

Kutanoglu e Sabuncuoglu em [Kutanoglu e Sabuncuoglu, 1999] analisam o desempenho de um conjunto de regras de prioridade na resolução do problema *Job-Shop* dinâmico, tendo como objectivo a minimização da soma dos atrasos pesados.

7.4 Notação utilizada

A tabela 7.1 apresenta a notação utilizada neste capítulo. No sequenciamento *Job-Shop* os *jobs* são constituídos por múltiplas operações, daí a necessidade de acrescentarmos alguma notação adicional relativamente à notação utilizada no sequenciamento com uma só máquina. Por exemplo, a operação O_{ijk} é constituída pelos parâmetros i, j e k, sendo j a máquina onde a operação k do *job* i é processada.

Tabela 7.1 Notação utilizada

n – número de *jobs*
m – número de máquinas
$J=\{J_1, J_2, ..., J_n\}$ – conjunto de *jobs*
$M=\{M_1, M_2, ..., M_n\}$ – conjunto de máquinas
j – máquina usada num dado momento ($j \in M$)
i – *job* a ser processado ($i \in J$)
k – operação de um *job*
O_{ijk} – operação k do *job* i na máquina j
r_{ijk} – instante em que a operação k do *job* i está disponível para processamento na máquina j
p_{ijk} – tempo de processamento da operação k do *job* i na máquina j
d_{ijk} – data de entrega da operação k do *job* i na máquina j
C_{ijk} – instante de finalização da operação k do *job* i na máquina j
$ITIO_{ijk}$ – intervalo dos tempos de início da operação k do *job* i na máquina j ($ITIO_{ijk} = [t^0_{ijk}, t^1_{ijk}]$)

t^0_{ijk} – tempo de início mais cedo da operação k do *job* i na máquina j

t^1_{ijk} – tempo de início mais tarde da operação k do *job* i na máquina j

C_i – instante de finalização do *job* i ($C_i = r_i + \sum_{j=1}^{m} \left(r_{ijk} + p_{ijk} \right)$)

P_i – tempo total de processamento do *job* i ($P_i = \sum_{j=1}^{m} p_{ijk}, \forall\, k \in N$)

A figura 7.1 apresenta um exemplo de uma rede de precedências, onde é possível visualizar o intervalo dos tempos de início das operações (*ITIO*), o tempo de processamento das operações *p*, o *job i*, as máquinas j_1, j_2 e j_3 e a respectiva precedência entre as operações.

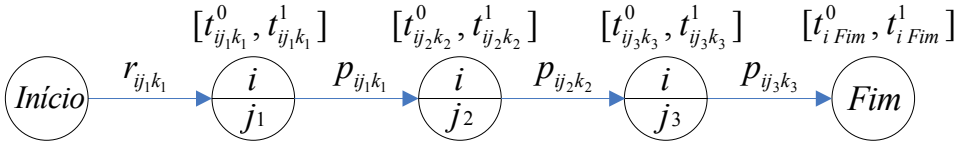


Figura 7.1 Exemplo de uma rede de precedências.

Como mostra a figura 7.1, para uma operação sem operações precedentes verifica-se a seguinte condição: $t^0_{ij_1k_1} = r_{ij_1k_1}$. Assim, o intervalo de tempo de início em que uma operação k_1 pode começar, $ITIO_{ij_1k_1}$, é determinado da seguinte forma:

$$ITIO_{ij_1k_1} = [t^0_{ij_1k_1}, t^1_{ij_1k_1}] = [r_{ij_1k_1}, t^1_{ij_1k_1}]. \tag{7.1}$$

O tempo de início mais cedo da operação k do *job* i na máquina j (t^0_{ijk}) corresponde ao instante de tempo a partir do qual o processamento da operação k pode ser iniciado. O tempo de início mais tarde da operação k do *job* i na máquina j (t^1_{ijk}) corresponde ao instante de tempo em que o processamento da operação k deve ser iniciado para que esta seja concluída na data de entrega prevista. Considerando a figura 7.1, os tempos de início mais cedo e os tempos de início mais tarde são determinados da seguinte forma:

$$
\begin{aligned}
ITIO_{ij_1k_1} &= [t^0_{ij_1k_1}, t^1_{ij_1k_1}] = [r_{ij_1k_1}, t^1_{ij_2k_2} - p_{ij_1k_1}] \\
ITIO_{ij_2k_2} &= [t^0_{ij_2k_2}, t^1_{ij_2k_2}] = [t^0_{ij_1k_1} + p_{ij_1k_1}, t^1_{ij_3k_3} - p_{ij_2k_2}] \\
ITIO_{ij_3k_3} &= [t^0_{ij_3k_3}, t^1_{ij_3k_3}] = [t^0_{ij_2k_2} + p_{ij_2k_2}, t^1_{i\,Fim} - p_{ij_3k_3}] \\
ITIO_{i\,Fim} &= [t^0_{i\,Fim}, t^1_{i\,Fim}] = [t^0_{ij_3k_3} + p_{ij_3k_3}, d_i]
\end{aligned}
\tag{7.2}
$$

Quando não são atribuídas datas de entrega aos *jobs* (d_i), deveremos considerar $t^1_{i\,Fim} = \max(t^0_{1\,Fim}, t^0_{2\,Fim}, ..., t^0_{n\,Fim})$, sendo n o número de *jobs*.

Consideremos agora a figura 7.2. Esta, apresenta um exemplo genérico de uma rede de precedências de um *job* com operações concorrentes (k_1 e k_2). As operações k_1 e k_2 são processadas em simultâneo nas máquinas j_1 e j_2, o que significa que estas operações não possuem qualquer relação de precedência entre si, podendo ser processadas ao mesmo tempo em máquinas diferentes (j_1 e j_2).

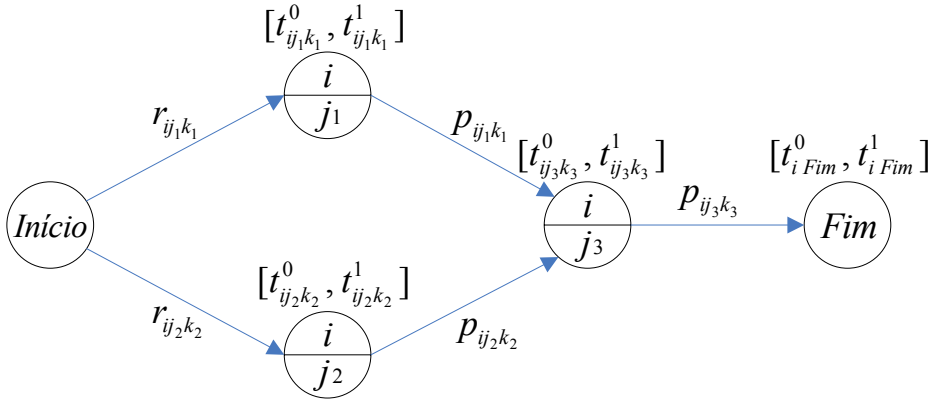


Figura 7.2 Rede de precedências de um *job* com operações concorrentes.

Considerando a figura 7.2, os intervalos dos tempos de início das operações são determinados da seguinte forma:

$$
\begin{aligned}
ITIO_{ij_1k_1} &= [t^0_{ij_1k_1}, t^1_{ij_1k_1}] = [r_{ij_1k_1}, t^1_{ij_3k_3} - p_{ij_1k_1}] \\
ITIO_{ij_2k_2} &= [t^0_{ij_2k_2}, t^1_{ij_2k_2}] = [r_{ij_2k_2}, t^1_{ij_3k_3} - p_{ij_2k_2}] \\
ITIO_{ij_3k_3} &= [t^0_{ij_3k_3}, t^1_{ij_3k_3}] = [t^0_{ij_1k_1} + p_{ij_1k_1}, t^1_{i\,Fim} - p_{ij_3k_3}] \cap [t^0_{ij_2k_2} + p_{ij_2k_2}, t^1_{i\,Fim} - p_{ij_3k_3}] \\
ITIO_{i\,Fim} &= [t^0_{i\,Fim}, t^1_{i\,Fim}] = [t^0_{ij_3k_3} + p_{ij_3k_3}, d_i]
\end{aligned}
\tag{7.3}
$$

Num *job* com operações concorrentes não é possível iniciar uma operação enquanto as operações precedentes não estiverem todas concluídas. Assim, para a figura 7.2, o intervalo dos tempo de início da operação $O_{ij_3k_3}$ é o resultado da intersecção de $[t^0_{ij_1k_1} + p_{ij_1k_1}, t^1_{i\,Fim} - p_{ij_3k_3}]$ com $[t^0_{ij_2k_2} + p_{ij_2k_2}, t^1_{i\,Fim} - p_{ij_3k_3}]$.

7.5 Modelo Proposto para os Problemas de Sequenciamento *Job-Shop*

O modelo proposto para a resolução de problemas de sequenciamento do tipo *Job-Shop* consiste em decompor o problema *Job-Shop* em vários problemas de uma só máquina. As soluções obtidas nos problemas de uma só máquina serão depois incorporadas no problema principal. Mas, a integração das várias soluções óptimas obtidas em cada um dos problemas de uma só máquina poderá conduzir a uma solução inválida ou inexequível para o problema original. Assim, é necessário ajustar as diferentes soluções obtidas para cada uma das máquinas no problema principal.

A figura 7.3 apresenta a arquitectura do modelo de sequenciamento desenvolvido no âmbito deste capítulo. Este modelo foi acrescentado ao *Hybrid and Flexible Genetic Algorithm* (HybFlexGA), desenvolvido e apresentado no capítulo cinco. Como mostra a figura 7.3, a arquitectura proposta está estruturada em três módulos principais: o módulo da interface gráfica, o módulo de pré-processamento e módulo de sequenciamento com AG.

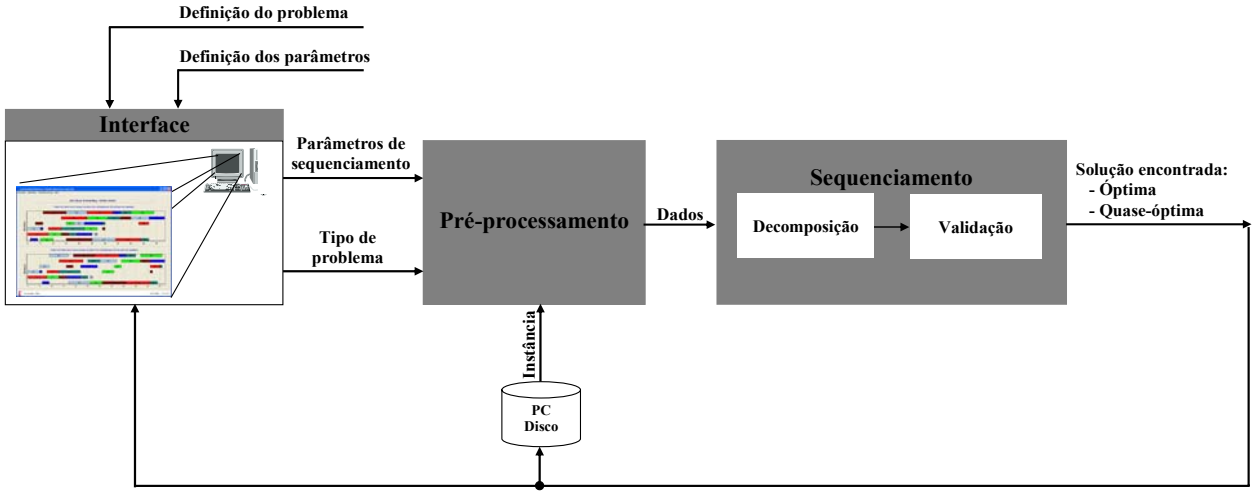
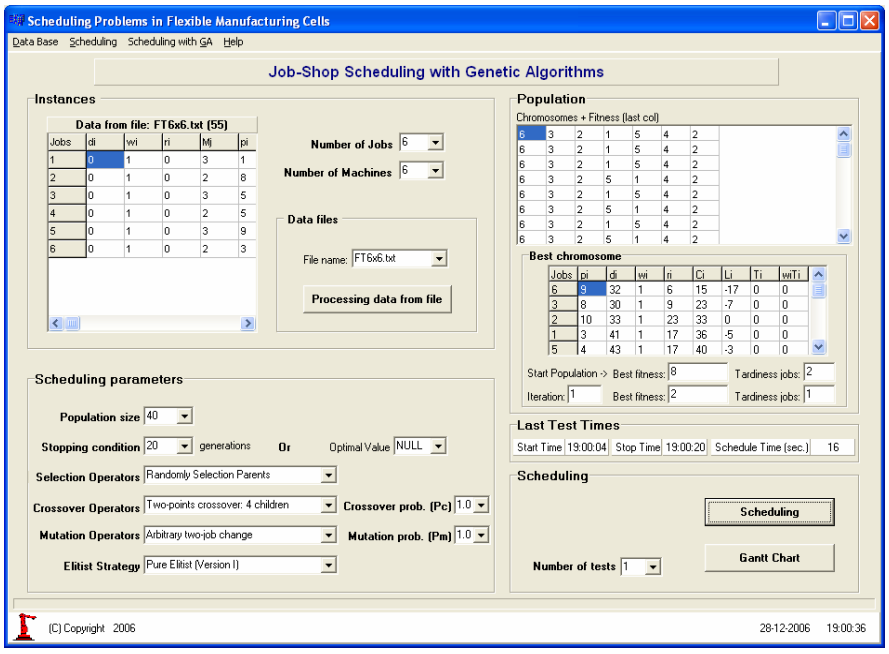
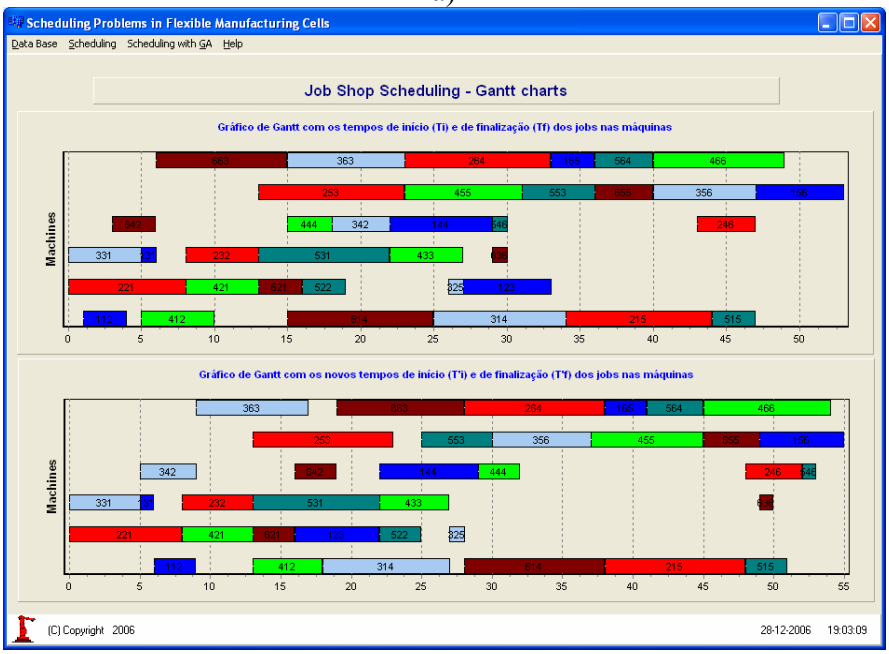


Figura 7.3 Arquitectura proposta para os problemas de sequenciamento do tipo *Job-Shop*.

O módulo de interface com o utilizador é determinante para o sucesso dum sistema de sequenciamento. Assim, esta interface deve ser amigável e dinâmica de modo a permitir aceder e manipular facilmente a informação relativa ao plano de sequenciamento, aos *jobs*, aos operadores genéticos, etc.. A interface desenvolvida permite a interactividade entre o utilizador e o módulo de sequenciamento, permitindo uma fácil introdução de dados e a visualização das soluções encontradas. A figura 7.4 mostra a interface desenvolvida.



a)



b)

Figura 7.4 Interface desenvolvida:
a) Sequenciamento *Job-Shop*
b) Visualização dos gráficos de *Gantt*

O módulo de pré-processamento trata a informação vinda do módulo da interface com o utilizador, que inclui o tipo de problema e os parâmetros de sequenciamento necessários ao módulo seguinte – módulo de sequenciamento. Assim, o módulo de pré-processamento trata a informação de entrada do sistema, como a definição da instância do problema e a parametrização do AG utilizado no módulo seguinte. A instância do problema de sequenciamento pode ser gerada aleatoriamente ou a partir dum ficheiro do disco do PC, como mostra a figura 7.3.

O módulo de sequenciamento tem como objectivo principal encontrar a solução óptima do problema de sequenciamento. Quando a solução óptima não é encontrada, o algoritmo devolve a melhor solução obtida (solução satisfatória, também designada, muitas vezes, de quase-óptima). A figura 7.5 mostra o algoritmo de sequenciamento proposto e implementado neste módulo. O 1º, 2º e 3º passos têm como objectivo decompor o problema *Job-Shop* em vários problemas de uma só máquina e encontrar as respectivas sequências óptimas ou quase-óptimas. A integração destas sequências no problema principal é feita no 4º passo, enquanto a verificação da exequibilidade da solução obtida é realizada no 5º passo. No 3º passo, foi utilizado o algoritmo genético desenvolvido no capítulo seis (ver figura 6.22 do capítulo 6).

Como mostra a figura 7.3, o módulo de sequenciamento está dividido em dois submódulos:

- Submódulo de decomposição: tem como objectivo obter uma solução para o problema *Job-Shop*, decompondo o problema original em vários problemas de uma só máquina. Como mostra a figura 7.5, o 1º, 2º, 3º e 4º passos do algoritmo de sequenciamento pertencem a este submódulo.
- Submódulo de validação: tem como objectivo verificar a exequibilidade da solução encontrada. A este submódulo pertence o 5º passo do algoritmo de sequenciamento, como é possível verificar na figura 7.5.

7.5.1 Submódulo de decomposição

No submódulo de decomposição, o problema original de sequenciamento é decomposto em vários problemas de uma só máquina. Seguidamente, é aplicado o AG a cada um dos problemas de uma só máquina, com o objectivo de encontrar as sequências óptimas (ou quase-óptimas) para cada uma das máquinas. Por último, as soluções encontradas pelo AG são integradas no problema principal.

Decomposição

1º Passo
Determinação do intervalo dos tempos de início de cada operação ($ITIO_{iik}$), em que as operações deverão ter início de modo a que as datas de entrega dos *jobs* sejam cumpridas.

2º Passo
Decomposição do problema original em vários problemas de uma só máquina.

3º Passo
Aplicação do AG a cada um dos problemas de uma só máquina.

4º Passo
Integração das soluções obtidas no problema principal.

Validação

5º Passo
Verificação da exequibilidade da solução, de acordo com o seguinte algoritmo:

Início
Repete:
A solução é exequível?
Se sim **então**
Foi encontrada a solução óptima?
Se sim **então**
Termina e devolve a solução encontrada
Senão
Enquanto (houver sequências alternativas)
Aplica o Mecanismo de Sequências Alternativas (MSA) às máquinas
Volta ao **Repete**
FimEnquanto
Devolve a melhor solução encontrada
FimSe
Senão
Aplica o Mecanismo de Coordenação das Operações (MCO) dos *jobs*
Volta ao **Repete**
FimSe
FimInício

Figura 7.5 Algoritmo de sequenciamento para os problemas do tipo *Job-Shop*.

7.5.2 Submódulo de validação

No submódulo de validação são analisadas todas as sequências encontradas pelo submódulo de decomposição com o fim de verificar se a solução encontrada é exequível. Caso a solução encontrada não seja exequível, o submódulo de validação coordenará as actividades das operações nas máquinas, tendo sempre em atenção as restrições de precedência definidas entre as operações de cada *job* e os tempos de ocupação das máquinas.

O submódulo de validação tem como principal objectivo verificar a exequibilidade da solução encontrada no 4º passo do submódulo de decomposição. Caso a solução encontrada seja exequível e óptima, o algoritmo termina e a solução encontrada será utilizada na construção do diagrama de *Gantt*. Se a solução encontrada não for exequível, o submódulo de validação irá coordenar as operações dos *jobs* nas máquinas, atendendo sempre às restrições de precedência entre as operações de cada *job* e os respectivos tempos de ocupação das máquinas.

Sempre que o submódulo de decomposição originar uma solução não exequível, serão aplicados ao nível do submódulo de validação mecanismos de sequenciamento com o intuito de, em primeiro lugar, tornar a solução exequível e, em segundo lugar, encontrar a sequência óptima (ou quase óptima) do problema em análise. Nas subsecções a seguir encontra-se uma descrição detalhada dos mecanismos desenvolvidos e implementados ao nível do submódulo de validação.

7.5.2.1 Mecanismo de Coordenação das Operações

O Mecanismo de Coordenação das Operações (MCO) é utilizado sempre que o submódulo de decomposição origina uma solução não exequível. Este mecanismo consiste em sequenciar as operações dos *jobs* nas máquinas de acordo com as sequências encontradas pelo AG no 3º passo. Por vezes, o sequenciamento das operações nas máquinas originam espaços livres (períodos mortos) entre operações. Estes espaços livres resultam, forçosamente, da coordenação dos tempos de ocupação das máquinas e das relações de precedência entre as operações executadas nas máquinas. Assim, e sempre que possível, as operações são sequenciadas para trás com o objectivo de se preencherem esses espaços livres, podendo esses sequenciamentos ser de dois tipos:

- Sequenciamento para trás sem arrastamento das operações a jusante;
- Sequenciamento para trás com arrastamento das operações a jusante.

Sempre que o espaço livre for maior ou igual ao tempo de processamento (p_{ijk}) da operação a sequenciar estamos perante um sequenciamento para trás sem arrastamento das operações que se encontram a jusante do espaço livre. Estes sequenciamentos, sempre que possível, devem ser tidos em conta, uma vez que melhoram sempre o resultado da solução final.

Os sequenciamentos para trás com arrastamento das operações a jusante ocorrem quando o espaço livre é menor que o tempo de processamento (p_{ijk}) da operação a sequenciar. Assim, o espaço livre ao ser preenchido por uma operação vai originar um arrastamento das operações que se encontram a jusante. Este arrastamento não só se verifica na máquina onde é preenchido o espaço livre, mas também pode ocorrer nas restantes máquinas devido à relação de precedência entre as operações. Desta forma, o sequenciamento para trás com arrastamento pode não contribuir para uma melhoria do resultado da solução final. Assim, sempre que ocorre um sequenciamento deste tipo é feita uma avaliação à solução encontrada, a fim de se verificar se este contribuiu, ou não, para uma melhoria da solução final. Sempre que há um agravamento da

solução final a operação é sequenciada para a frente, ou seja, ficando imediatamente a seguir à última operação sequenciada na máquina.

Como conclusão podemos dizer que, os sequenciamentos para trás sem arrastamento de operações contribuem sempre para uma melhoria da solução final. Os sequenciamentos para trás com arrastamento podem ou não contribuir para uma melhoria da solução final.

A seguir, apresentamos os algoritmos das funções desenvolvidas e implementadas no MCO. As referidas funções são:

– *SequenciamentoParaFrenteParaTrasComOuSemArrastamento()*, figura 7.6;

– *ProcuraEspacosLivresParaTras()*, figura 7.7;

– *PreenchimentoEspacosLivresParaTrasSemArrastamento()*, figura 7.8;

– *PreenchimentoEspacosLivresParaTrasComArrastamento()*, figura 7.9;

– *EspacoLivrePreenchidoParaTrasTrueOrFalse()*, figura 7.10;

– *OrdenarMatriz_NovosTiTf()*, figura 7.11.

```
Início
   Para i=0 até TotalMaquinas fazer
      Fazer
         Na sequência óptima obtida pelo AG, determinar a operação a sequenciar
         Se (operação anterior sequenciada) então
            Anula a operação a sequenciar na sequência óptima obtida pelo AG
            OperAnteriorSequenciada←verdadeira
         SeNão
            Repõe a operação a sequenciar na sequência óptima obtida pelo AG
            Avança para a operação seguinte
         FimSe

         Para i=0 até TotalJobsTodasMaquinas e OperAnteriorSequenciada fazer
            NumMaquina←Maquina
            EspaçoLivrePreechido←falso
            ri←instante de chegada da operação a sequenciar
            pi←tempo de processamento da operação a sequenciar
            ProcuraEspacosLivresParaTras(NumMaquina)
            PreenchimentoEspacosLivresParaTrasSemArrastamento(TotalEspacosLivres, pi)
            PreenchimentoEspacosLivresParaTrasComArrastamento(TotalEspacosLivres, pi)
            EspacoLivrePreenchidoParaTrasTrueOrFalse()
            Operação sequenciada
            Se (espaço livre preenchido=falso) então TfComparacao_Maq=novoTf_Maq
            Se (espaço livre preenchido=verdadeiro) então OrdenarMatriz_NovosTiTf(NumMaquina)
         FimPara
      Enquanto (operações para sequenciar)

   //Determinar o makespan
   Se (makespan melhorou) então makespanBest←makespanActual
FimInício
```

Figura 7.6 Algoritmo da função *SequenciamentoParaFrenteParaTrasComOuSemArrastamento()*

```
Início
   Espaço livre ←0
   Para i=0 até TotalJobsTodasMaquinas fazer
      Se (encontrou espaço livre) então
         Determina o limite inferior do espaço livre
         Determina o limite superior do espaço livre
         Determina o tamanho do espaço livre
         Espaço livre ← Espaço livre+1;
      FimSe
   FimPara
FimInício
```

Figura 7.7 Algoritmo da função *ProcuraEspacosLivresParaTras()*.

```
Início
   Se (houver espaços livres e espaço livre >= pi) então
      Determina o novoTi_Maq
      Determina o novoTf_Maq
      EspaçoLivrePreenchidoSemArrastamento ← verdadeiro
   FimSe
FimInício
```

Figura 7.8 Algoritmo da função *PreenchimentoEspacosLivresParaTrasSemArrastamento()*.

```
Início
   Se (houver espaços livres e espaço livre < pi) então
      Procura o maior espaço livre
      Determina o novoTi_Maq
      Determina o novoTf_Maq
      EspaçoLivrePreenchidoComArrastamento ← verdadeiro
   FimSe
FimInício
```

Figura 7.9 Algoritmo da função *PreenchimentoEspacosLivresParaTrasComArrastamento()*.

```
Início
   Se (EspaçoLivrePreenchidoSemArrastamento = verdadeiro ou
       EspaçoLivrePreenchidoComArrastamento = verdadeiro) então
      EspaçoLivrePreenchido ← verdadeiro
   FimSe
   Se (EspaçoLivrePreenchidoSemArrastamento = falso e
       EspaçoLivrePreenchidoComArrastamento = falso) então
      EspaçoLivrePreenchido ← falso
   FimSe
FimInício
```

Figura 7.10 Algoritmo da função *EspacoLivrePreenchidoParaTrasTrueOrFalse()*.

```
Início
   Ordena a matriz dos novosTiTf por ordem crescente
   Se (EspaçoLivrePreenchidoComArrastamento = verdadeiro)
      Procura as operações arrastadas na máquina onde se deu o arrastamento
      Procura as operações seguintes nas restantes máquinas
      ReSequencia as operações que sofreram arrastamento
   FimSe
FimInício
```

Figura 7.11 Algoritmo da função *OrdenarMatriz_NovosTiTf()*.

7.5.2.2 Mecanismo de Sequências Alternativas

O Mecanismo de Sequências Alternativas (MSA) é utilizado sempre que no 3º passo o AG encontra mais do que uma sequência de igual valor para a mesma máquina. Para cada máquina, o MSA utiliza as melhores sequências geradas pelo AG como soluções alternativas, com vista ao melhoramento da solução final do problema. O MSA faz a integração de cada uma dessas sequências no problema principal e verifica se houve uma melhoria da solução global. Quando uma determinada sequência contribui para uma melhoria da solução global do problema, esta, é considerada, caso contrário é rejeitada. Para ilustrarmos melhor o princípio de funcionamento do MSA, consideremos o sequenciamento de 3 *jobs* e 3 máquinas apresentado na figura 7.12.

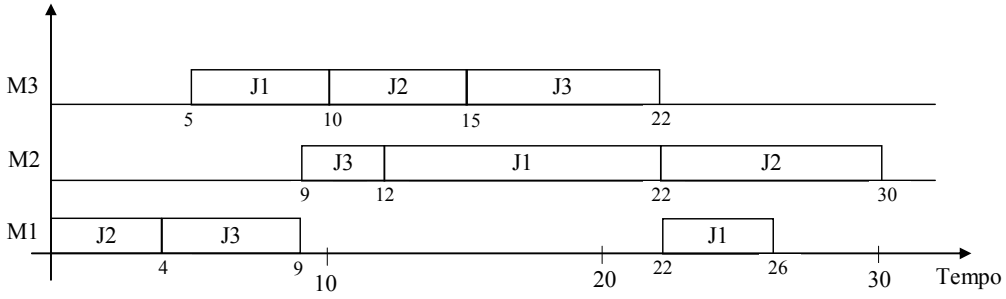


Figura 7.12 Sequenciamento sem aplicação do MSA.

Relativamente à máquina 1, consideremos que o J_1 tem um instante de chegada r_1=20, o J_2 tem um instante de chegada r_2=0 e o J_3 apresenta um instante de chegada r_3=0. A tabela 7.2 apresenta as sequências possíveis para esta máquina, onde é possível verificar que existem duas sequências com o mesmo valor de C_{max}=24. A primeira sequência (J_2, J_3, J_1) foi utilizada no sequenciamento da figura 7.12, apresentando este um *makespan* de 30 unidades de tempo. E se fosse usada a sequência alternativa (J_3, J_2, J_1) através da aplicação do MSA, o *makespan* melhoraria? A resposta a esta pergunta encontra-se no gráfico de *Gantt* da figura 7.13. Como mostra esta figura, houve uma melhoria do *makespan* em duas unidades de tempo relativamente ao sequenciamento apresentado na figura 7.12. Assim, podemos concluir que da aplicação do MSA poderá resultar um melhoramento da solução global do problema.

Tabela 7.2 Sequências possíveis para a máquina 1

Sequências			C_{max}
J_2	J_3	J_1	24
J_3	J_2	J_1	24
J_3	J_1	J_2	28
J_2	J_1	J_3	29
J_1	J_2	J_3	33
J_1	J_3	J_2	33

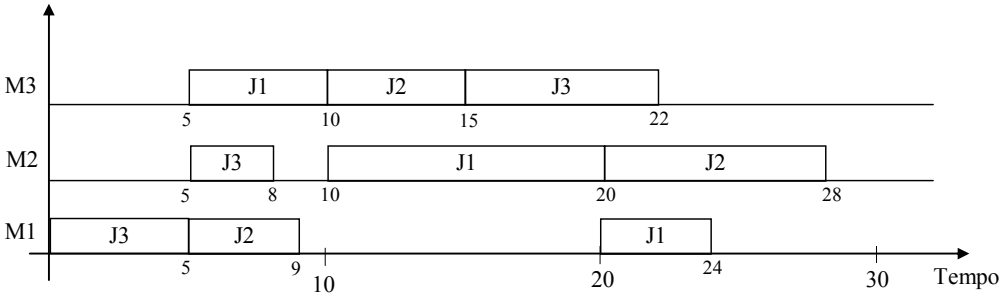


Figura 7.13 Sequenciamento após a aplicação do MSA.

7.6 Ilustração do algoritmo de sequenciamento proposto

A finalidade desta secção é apresentar um conjunto de exemplos de forma a ilustrar a aplicação do algoritmo de sequenciamento proposto.

7.6.1 Exemplo 1

Consideremos o problema de sequenciamento do tipo 5/3/G/$C_{\max}$, adaptado de Dileep [Dileep, 1996]. A tabela 7.3 apresenta, para cada *job*, a sequência das operações nas máquinas e o tempo de processamento requerido em cada máquina. Por exemplo, o *job* 1 requer as máquinas 1 e 3, nesta ordem, e 1 unidade de tempo de processamento na máquina 1, e 2 unidades de tempo de processamento na máquina 3. Suponhamos também que todos os *jobs* estão disponíveis para processamento no instante de tempo t=0, i.e., r_{ijk}=0 para todos os *jobs*. O critério de desempenho que se pretende optimizar é a minimização do tempo total de finalização da produção $C_{\max}$ (*completion time* ou *makespan*).

Tabela 7.3 Dados do exemplo 1

Jobs	**Sequência das operações nas máquinas**	**Tempos de processamento (p_{ijk})**
1	1,3	1,2
2	2,3	3,1
3	3,1,2	2,1,1
4	1,2,3	3,2,1
5	3,1	2,1

1º Passo: Determinação do intervalo dos tempos de início de cada operação ($ITIO_{ijk}$)

O intervalo dos tempos de início para cada operação indica o tempo em que a operação deve ser iniciada para que a data de entrega do *job* seja cumprida. O intervalo dos tempos de início da operação O_{ijk} é dado por $ITIO_{ijk} = [t^0_{ijk}, t^1_{ijk}]$, sendo t^0_{ijk} o tempo de início mais cedo da operação O_{ijk} e t^1_{ijk} o tempo de início mais tarde da operação O_{ijk}.

Por exemplo, para o *job* 1 os intervalos dos tempos de início de cada operação ($ITIO_{ijk}$) são:

$$\begin{aligned} ITIO_{111} &= [r_{111},\, t^1_{132} - p_{111}] = [0,\, 3] \\ ITIO_{132} &= [t^0_{111} + p_{111},\, t^1_{1\,Fim} - p_{132}] = [1,\, 4] \\ ITIO_{1\,Fim} &= [t^0_{132} + p_{132},\, d_1] = [3,\, 6] \end{aligned} \tag{7.4}$$

Como no exemplo em questão não foram atribuídas datas de entrega aos *jobs* (d_i), temos que $t^1_{i\,Fim} = \max(t^0_{1\,Fim}, ..., t^0_{n\,Fim})$. Assim, para todos os *jobs*, teremos que calcular em primeiro lugar $t^0_{i\,Fim}$ para de seguida obtermos $t^1_{i\,Fim}$ (o máximo $t^0_{i\,Fim}$ encontrado nos vários *jobs*). No exemplo em questão, $t^1_{i\,Fim} = 6$ (ver figura 7.14).

A figura 7.14 apresenta a rede de todos os *jobs* do exemplo 1, sendo possível visualizar a precedência dos *jobs*, os intervalos dos tempos de início das operações ($ITIO_{ijk}$) e os tempos de processamento das operações nas máquinas (p_{ijk}).

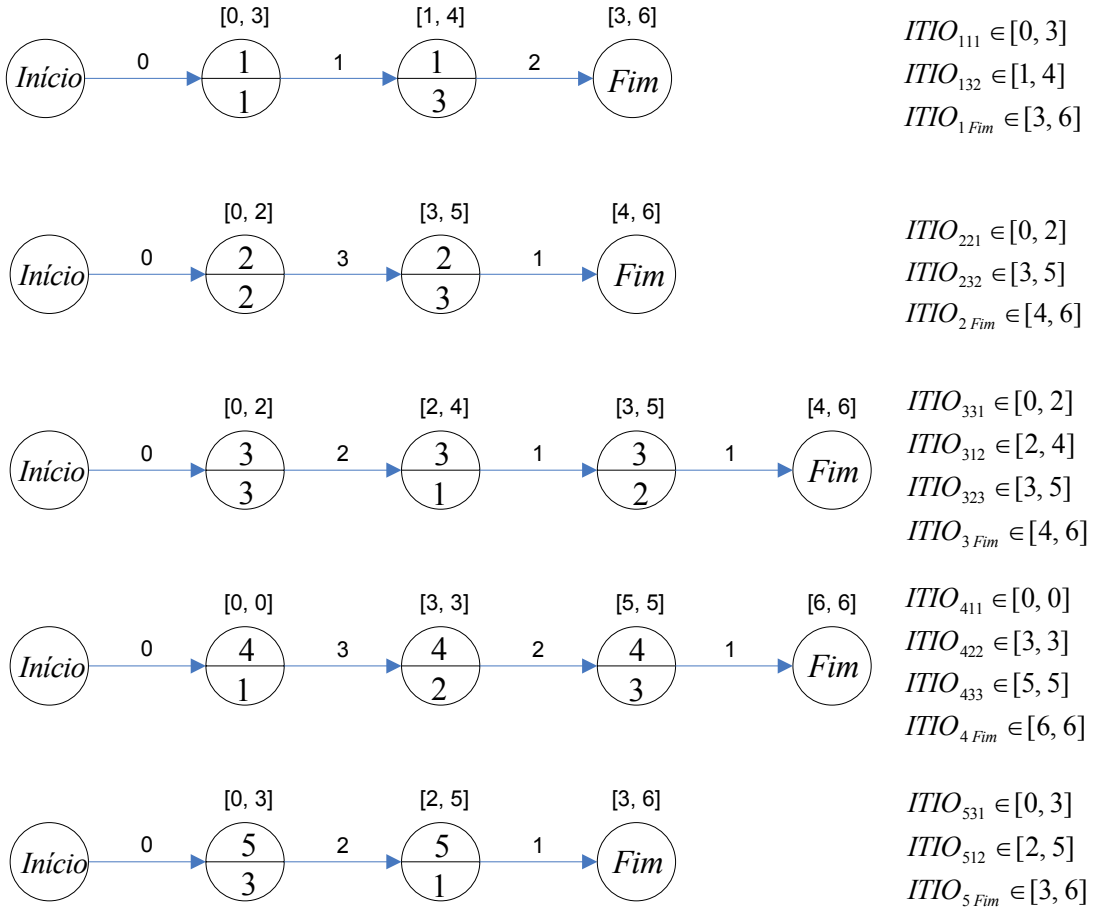


Figura 7.14 Rede de precedências de todos os *jobs*.

2º Passo: Decomposição do problema original em vários problemas de uma só máquina

O objectivo principal deste passo é decompor o problema original em vários problemas de uma só máquina. Aqui, e uma vez que os problemas de uma só máquina são do tipo uni-operação, usaremos a notação r_i, p_i e d_i para representar o instante de chegada, o tempo de processamento e a data de entrega do *job i* (esta notação foi a utilizada nos capítulos 4 e 5).

Os instantes de chegada dos *jobs* r_i correspondem ao valor do limite inferior do intervalo de tempo de início mais cedo t^0_{ijk} das operações, isto é, $r_i = t^0_{ijk}$. O tempo de processamento p_i do *job* é igual ao tempo de processamento da respectiva operação p_{ijk}, isto é, $p_i = p_{ijk}$. As datas de entrega d_i correspondem aos instantes de finalização das operações C_{ijk}, ou seja, $d_i = C_{ijk}$.

Relativamente ao *job* 1, o instante de finalização da última operação é 6 (máximo $t^0_{i\,Fim}$ encontrado nos vários *jobs*), o instante de finalização da penúltima operação é determinado subtraindo ao C_{ijk} da última operação o respectivo tempo de processamento, e assim sucessivamente até à 1ª operação. Assim, para o *job* 1 temos: C_{111}=4 e C_{132}=6. Os instantes de finalização também podem ser determinados a partir dos intervalos dos tempos de início de cada operação ($ITIO_{ijk}$) da seguinte forma: $C_{111} = t^1_{132}$ e $C_{132} = t^1_{1\,Fim}$ (ver figura 7.14). Assim, a tabela 7.4 apresenta a decomposição do problema original em três problemas de uma só máquina.

Tabela 7.4 Dados do problema de uma só máquina

Máquina 1				**Máquina 2**				**Máquina 3**			
Jobs	r_i	p_i	d_i	***Jobs***	r_i	p_i	d_i	***Jobs***	r_i	p_i	d_i
111	0	1	4	221	0	3	5	132	1	2	6
312	2	1	5	323	3	1	6	232	3	1	6
411	0	3	3	422	3	2	5	331	0	2	4
512	2	1	6					433	5	1	6
								531	0	2	5

3º Passo: Aplicação do AG a cada um dos problemas de uma só máquina

Aqui, aplicamos o AG a cada um dos problemas de uma só máquina obtidos no passo anterior, com o objectivo de encontrar a solução óptima ou quase-óptima para cada um desses problemas. Lembramos que, o critério de desempenho que se pretende optimizar neste exemplo é a minimização do tempo total de finalização da produção $C_{\max}$ (*makespan*). Assim, após a aplicação do AG a cada um dos problemas de uma só máquina, obtidos no 2º passo, obtiveram-se as seguintes soluções:

Máquina 1: S_{opt}=(111, 411, 312, 512)

Máquina 2: S_{opt}=(221, 422, 323)

Máquina 3: S_{opt}=(331, 531, 132, 232, 433)

4º Passo: Integração das soluções obtidas no problema principal

A tabela 7.5 mostra os tempos de início T_i e os tempos de finalização T_f de cada *job* nas 3 máquinas. Com base nos dados da tabela 7.5 obtém-se o sequenciamento apresentado na figura 7.15.

Tabela 7.5 Tempos de início e de finalização dos *jobs* nas máquinas

Máquina 1			**Máquina 2**			**Máquina 3**		
Jobs	T_i	T_f	***Jobs***	T_i	T_f	***Jobs***	T_i	T_f
111	0	1	221	0	3	331	0	2
411	1	4	422	3	5	531	2	4
312	4	5	323	5	6	132	4	6
512	5	6				232	6	7
						433	7	8

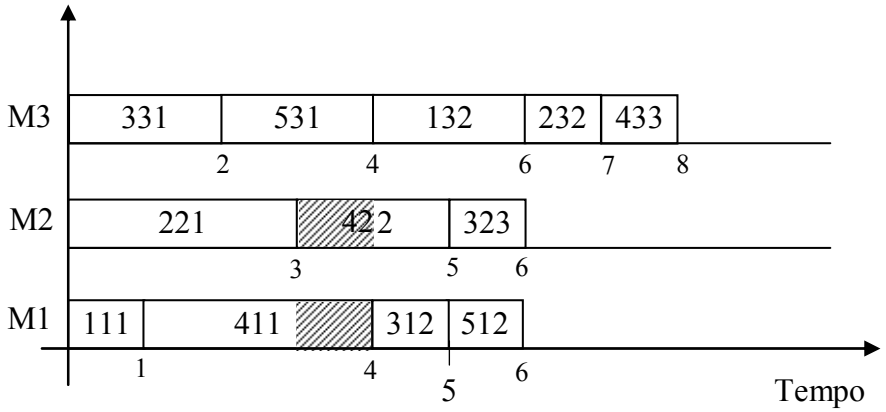


Figura 7.15 Sequenciamento resultante da integração das soluções obtidas no 3º passo.

5º Passo: Verificação da exequibilidade da solução

O sequenciamento apresentado na figura 7.15 não é exequível pelo facto da operação O_{411} se encontrar sobreposta à operação O_{422}. Na tabela 7.5, os tempos de início (T_i) e os tempos de finalização (T_f) de cada *job* foram determinados com base, unicamente, na sequência das operações realizadas nas máquinas obtida pelo AG, nos instantes de chegada (r_i) e nos tempos de processamento (p_i) dos *jobs*. Até ao 4º passo, o algoritmo não entra em linha de conta com a precedência imposta pela ocupação das máquinas.

Uma vez que o sequenciamento obtido na figura 7.15 não é exequível, é necessário proceder à coordenação das operações dos *jobs* nas várias máquinas. Consequentemente, a tabela 7.6 apresenta os "novos" tempos de início T'_i e de finalização T'_f de cada *job*, com o objectivo de tornar exequível o sequenciamento apresentado na figura 7.15. Estes valores resultam da aplicação do mecanismo de coordenação das operações (MCO) dos vários *jobs* nas máquinas. Por sua vez, a figura 7.16 apresenta o sequenciamento exequível, resultante dos tempos de início e de finalização (T'_i e T'_f) apresentados na tabela 7.6.

Tabela 7.6 Novos tempos de início e de finalização dos *jobs* nas máquinas

Máquina 1			**Máquina 2**			**Máquina 3**		
Jobs	***T'i***	***T'f***	***Jobs***	***T'i***	***T'f***	***Jobs***	***T'i***	***T'f***
111	0	1	221	0	3	331	0	2
411	1	4	422	4	6	531	2	4
312	4	5	323	6	7	132	4	6
512	5	6				232	6	7
						433	7	8

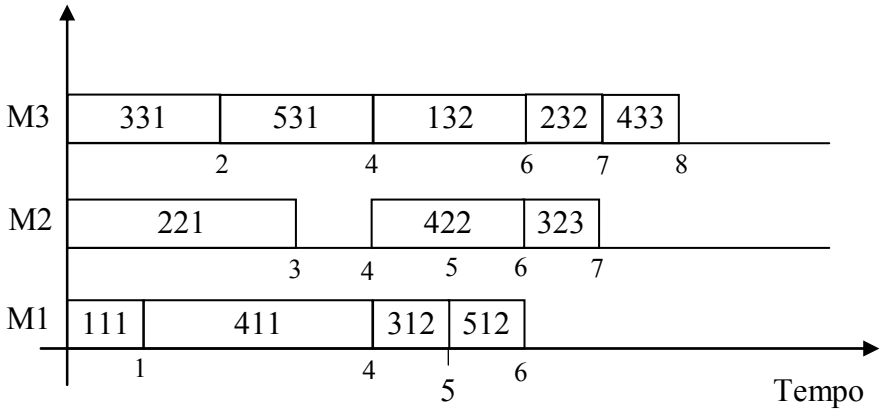


Figura 7.16 Sequenciamento resultante da aplicação do 5º passo do algoritmo.

O gráfico de *Gantt* apresentado na figura 7.16 mostra a solução válida do problema e o respectivo *makespan*, sendo, no exemplo apresentado, de 8 unidades de tempo. Também é possível visualizar no gráfico de *Gantt* que as máquinas 1 e 2 apresentam uma produtividade de 75% e a máquina 3 uma produtividade de 100%. O gráfico de *Gantt* mostra, também, a solução exequível: a sequência das operações dos *jobs* nas máquinas é a correcta e nenhum *job* é simultaneamente sequenciado em duas máquinas.

7.6.2 Exemplo 2

Consideremos o seguinte problema de sequenciamento do tipo 3/4/*G*/$C_{\max}$, adaptado de Pinedo [Pinedo, 1995]. A tabela 7.7 mostra, para cada *job*, a sequência das operações nas máquinas, os tempos de processamento requeridos em cada máquina e as datas de entrega dos *jobs*. Por exemplo, o *job* 1 requer as máquinas 1, 2 e 3, nesta ordem, e 10 unidades de tempo de processamento na máquina 1, 8 unidades de tempo de processamento na máquina 2 e 4 unidades de tempo de processamento na máquina 3. As datas de entrega dos *jobs* 1, 2 e 3 são 27, 33 e 18, respectivamente. O critério de desempenho que se pretende optimizar é a minimização do tempo total de finalização da produção $C_{\max}$ (*completion time* ou *makespan*).

Tabela 7.7 Dados do exemplo 2

Jobs	**Sequência das operações nas máquinas**	**Tempos de processamento (p_{ijk})**	**Datas de entrega (d_i)**
1	1,2,3	10, 8, 4	27
2	2,1,4,3	8, 3, 5, 6	33
3	1,2,4	4, 7, 3	18

1º Passo: Determinação do intervalo dos tempos de início de cada operação ($ITIO_{ijk}$)

O intervalo dos tempos de início da operação O_{ijk} é dado por $ITIO_{ijk} = [t^0_{ijk}, t^1_{ijk}]$, sendo t^0_{ijk} o tempo de início mais cedo da operação O_{ijk} e t^1_{ijk} o tempo de início mais tarde da operação O_{ijk}. Como neste exemplo foram atribuídas datas de entrega aos *jobs*, temos: $t^1_{1Fim} = 27$, $t^1_{2Fim} = 33$ e $t^1_{3Fim} = 18$.

Por exemplo, para o *job* 1 os intervalos dos tempos de início de cada operação ($ITIO_{ijk}$) são:

$$\begin{aligned} ITIO_{111} &= [r_{111}, t^1_{122} - p_{111}] = [0, 5] \\ ITIO_{122} &= [t^0_{111} + p_{111}, t^1_{133} - p_{122}] = [10, 15] \\ ITIO_{133} &= [t^0_{122} + p_{122}, t^1_{1\,Fim} - p_{133}] = [18, 23] \\ ITIO_{1\,Fim} &= [t^0_{133} + p_{133}, d_1] = [22, 27] \end{aligned} \tag{7.5}$$

A figura 7.17 apresenta a rede de precedências de todos os *jobs*, com indicação dos intervalos de tempos de início de cada operação ($ITIO_{ijk}$) e os tempos de processamento das operações nas máquinas (p_{ijk}).

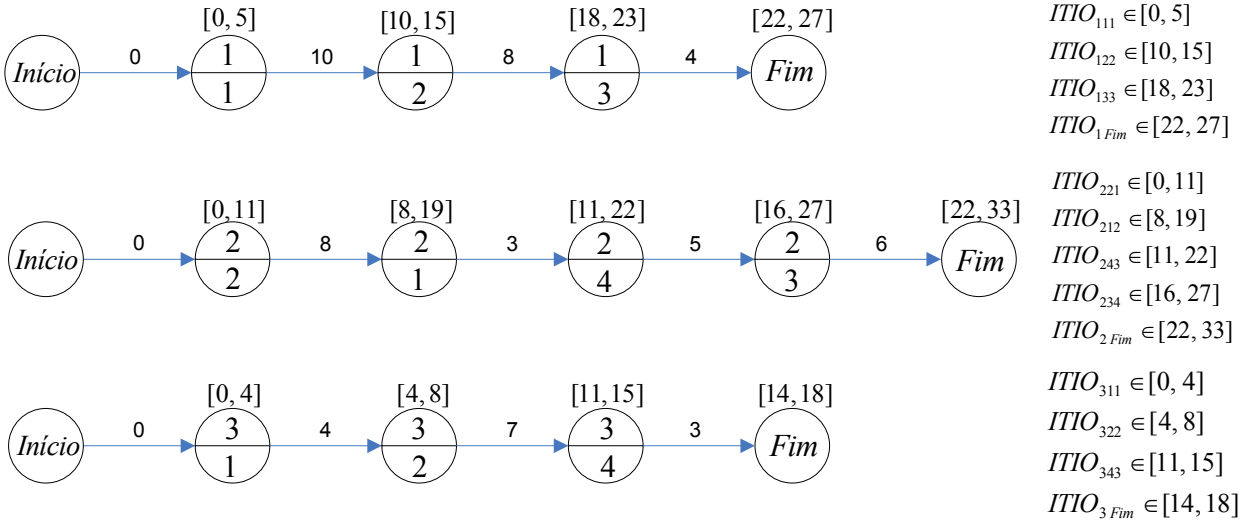


Figura 7.17 Rede de precedências de todos os *jobs*.

2º Passo: Decomposição do problema original em vários problemas de uma só máquina

Como já foi referido anteriormente, o objectivo principal deste passo é decompor o problema original em vários problemas de uma só máquina. Os instantes de chegada dos *jobs* r_i correspondem ao valor do limite inferior do intervalo de tempo de início mais cedo t^0_{ijk} das operações, isto é, $r_i = t^0_{ijk}$. O tempo de processamento p_i do *job* é igual ao tempo de processamento da respectiva operação p_{ijk}, isto é, $p_i = p_{ijk}$. As datas de entrega d_i correspondem aos instantes de finalização das operações C_{ijk}, ou seja, $d_i = C_{ijk}$.

Relativamente ao *job* 1, o instante de finalização da última operação corresponde à sua data de entrega, e o instante de finalização da 2ª operação é determinado subtraindo ao C_{ijk} da última operação o respectivo tempo de processamento, e assim sucessivamente até à 1ª operação. Assim, para o *job* 1 temos: C_{111}=15, C_{122}=23 e C_{133}=27. Os instantes de finalização também podem ser determinados a partir dos intervalos dos tempos de início de cada operação ($ITIO_{ijk}$) da seguinte forma: $C_{111} = t^1_{122}$, $C_{122} = t^1_{133}$ e $C_{133} = t^1_{1\,Fim}$ (ver figura 7.17). Assim, a tabela 7.8 apresenta a decomposição do problema original em quatro problemas de uma só máquina.

Tabela 7.8 Dados do problema de uma só máquina

Máquina 1				**Máquina 2**				**Máquina 3**				**Máquina 4**			
Jobs	r_i	p_i	d_i	***Jobs***	r_i	p_i	d_i	***Jobs***	r_i	p_i	d_i	***Jobs***	r_i	p_i	d_i
111	0	10	15	122	10	8	23	133	18	4	27	243	11	5	27
212	8	3	22	221	0	8	19	234	16	6	33	343	11	3	18
311	0	4	8	322	4	7	15								

3º Passo: Aplicação do AG a cada um dos problemas de uma só máquina

Aqui, aplicamos o AG a cada um dos problemas de uma só máquina obtidos no passo anterior, a fim de se encontrar a solução óptima ou quase-óptima para cada um desses problemas. Lembramos que, o critério de desempenho que se pretende optimizar neste exemplo é a minimização do tempo total de finalização da produção C_{max} (*makespan*). Assim, após a aplicação do AG a cada um dos problemas de uma só máquina, obtidos no 2º passo, obtiveram-se as seguintes soluções:

Máquina 1: S_{opt}=(111, 212, 311)

Máquina 2: S_{opt}=(221, 122, 322)

Máquina 3: S_{opt}=(133, 234)

Máquina 4: S_{opt}=(243, 343)

4º Passo: Integração das soluções obtidas no problema principal

A tabela 7.9 mostra os tempos de início T_i e os tempos de finalização T_f de cada *job* nas 4 máquinas. O sequenciamento apresentado na figura 7.18 foi obtido com base nos dados da tabela 7.9.

Tabela 7.9 Tempos de início e de finalização dos *jobs* nas máquinas

Máquina 1			**Máquina 2**			**Máquina 3**			**Máquina 4**		
Jobs	T_i	T_f	***Jobs***	T_i	T_f	***Jobs***	T_i	T_f	***Jobs***	T_i	T_f
111	0	10	221	0	8	133	18	22	243	11	16
212	10	13	122	10	18	234	22	28	343	16	19
311	13	17	322	18	25						

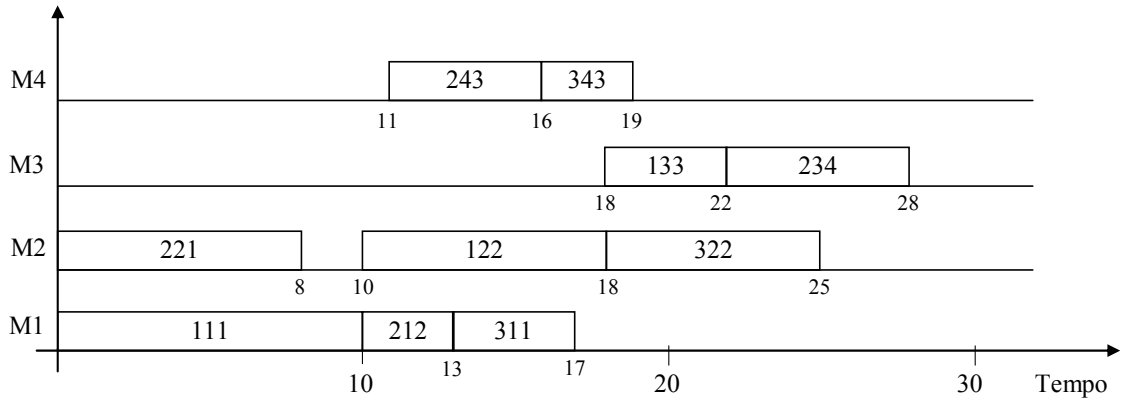


Figura 7.18 Sequenciamento resultante da integração das soluções obtidas no 3º passo.

5º Passo: Verificação da exequibilidade da solução

Como sabemos, o mesmo *job* não pode estar simultaneamente a ser processado em duas máquinas diferentes. Esta condição não se verifica no sequenciamento apresentado na figura 7.18 (a operação O_{212} encontra-se sobreposta à operação O_{243}), tornando este sequenciamento não exequível. No *job* 3 ocorre uma situação semelhante (ver figura 7.18).

O sequenciamento apresentado na figura 7.18 não é exequível, uma vez que os tempos de início T_i foram determinados com base, unicamente, nas datas de entrega dos *jobs* e respectivas sequências das operações nas máquinas. Até ao 4º passo, o algoritmo não entra em linha de conta com a precedência imposta pela ocupação das máquinas, resultante dos tempos de ocupação das mesmas.

Uma vez que o sequenciamento obtido na figura 7.18 não é exequível, é necessário proceder à coordenação das operações dos *jobs* nas várias máquinas. A tabela 7.10 apresenta os "novos" tempos de início T'_i e de finalização T'_f de cada *job*, de forma a tornar exequível o sequenciamento. Estes valores são obtidos através da aplicação do mecanismo de coordenação das operações (MCO) dos vários *jobs* nas máquinas. A figura 7.19 apresenta o sequenciamento exequível, resultante dos tempos de início e de finalização (T'_i e T'_f) indicados na tabela 7.10.

Tabela 7.10 Novos tempos de início e de finalização dos *jobs* nas máquinas

Máquina 1			**Máquina 2**			**Máquina 3**			**Máquina 4**		
Jobs	***T'i***	***T'f***	***Jobs***	***T'i***	***T'f***	***Jobs***	***T'i***	***T'f***	***Jobs***	***T'i***	***T'f***
111	0	10	221	0	8	133	18	22	243	13	18
212	10	13	122	10	18	234	22	28	343	25	28
311	13	17	322	18	25						

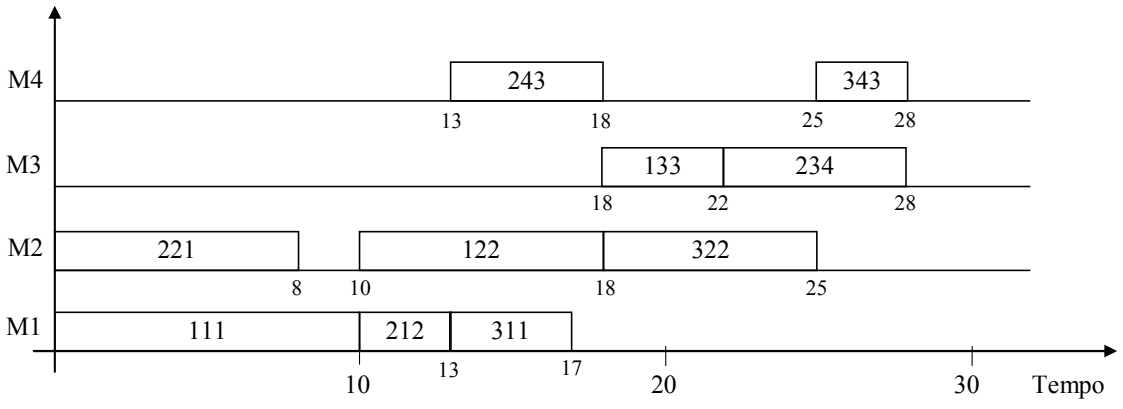


Figura 7.19 Sequenciamento resultante da aplicação do 5º passo do algoritmo.

O gráfico de *Gantt* mostra a solução válida do problema e o respectivo *makespan*, sendo, no exemplo apresentado, de 28 unidades de tempo. Também é possível visualizar no gráfico de *Gantt* a produtividade e o tempo morto de cada máquina. Por exemplo, a máquina 1 tem uma produtividade de 60,7% e um tempo morto de 39,3%. Por último, o gráfico de *Gantt* também mostra que o *job* 3 é o único *job* em atraso.

7.6.3 Exemplo 3

A instância utilizada neste exemplo é uma *benchmark* muito popular na comunidade científica, sendo utilizada com muita frequência por vários autores e investigadores nos seus trabalhos de investigação. Esta instância foi proposta por Fisher e Thompson [Fisher e Thompson, 1963] e é um problema de sequenciamento do tipo $6/6/G/C_{\max}$. O critério de desempenho que se pretende optimizar é a minimização do tempo total de finalização da produção $C_{\max}$ (*completion time* ou *makespan*). A solução óptima desta instância é 55, ou seja, *makespan*=55.

A tabela 7.11 apresenta, para cada *job*, a sequência das operações nas máquinas e o tempo de processamento requerido em cada máquina.

Tabela 7.11 Dados do exemplo 3

Jobs	**Sequência das operações nas máquinas**	**Tempos de processamento (p_{ijk})**
1	3,1,2,4,6,5	1,3,6,7,3,6
2	2,3,5,6,1,4	8,5,10,10,10,4
3	3,4,6,1,2,5	5,4,8,9,1,7
4	2,1,3,4,5,6	5,5,5,3,8,9
5	3,2,5,6,1,4	9,3,5,4,3,1
6	2,4,6,1,5,3	3,3,9,10,4,1

1º Passo: Determinação do intervalo dos tempos de início de cada operação ($ITIO_{ijk}$)

A figura 7.20 apresenta a rede de precedências de todos os *jobs*, onde é possível visualizar a precedência dos *jobs*, os intervalos dos tempos de início das operações ($ITIO_{ijk}$) e os tempos de processamento das operações nas máquinas (p_{ijk}). Por exemplo, para o *job* 1 os intervalos dos tempos de início de cada operação ($ITIO_{ijk}$) são: $ITIO_{131} \in [0, 21]$, $ITIO_{112} \in [1, 22]$, $ITIO_{123} \in [4, 25]$, $ITIO_{144} \in [10, 31]$, $ITIO_{165} \in [17, 38]$, $ITIO_{156} \in [20, 41]$, $ITIO_{1Fim} \in [26, 47]$.

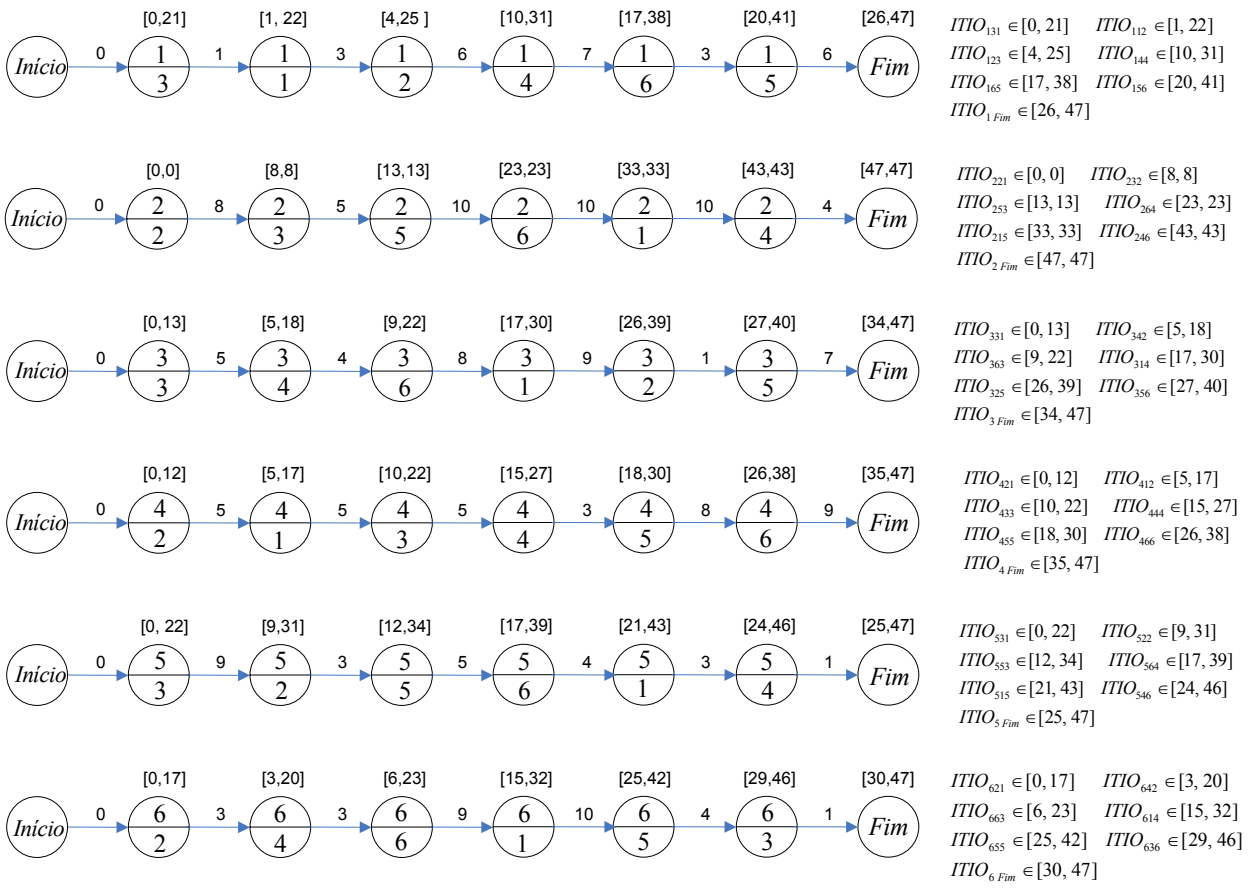


Figura 7.20 Rede de precedências de todos os *jobs*.

2º Passo: Decomposição do problema original em vários problemas de uma só máquina

A tabela 7.12 apresenta a decomposição do problema original em seis problemas de uma só máquina.

Tabela 7.12 Dados do problema de uma só máquina

Máquina 1						**Máquina 2**						**Máquina 3**					
Jobs	***ijk***	r_i	p_i	d_i	w_i	***Jobs***	***ijk***	r_i	p_i	d_i	w_i	***Jobs***	***ijk***	r_i	p_i	d_i	w_i
1	112	1	3	25	1	1	123	4	6	31	1	1	131	0	1	22	1
2	215	33	10	43	1	2	221	0	8	8	1	2	232	8	5	13	1
3	314	17	9	39	1	3	325	26	1	40	1	3	331	0	5	18	1
4	412	5	5	22	1	4	421	0	5	17	1	4	433	10	5	27	1
5	515	21	3	46	1	5	522	9	3	34	1	5	531	0	9	31	1
6	614	15	10	42	1	6	621	0	3	20	1	6	636	29	1	47	1
Máquina 4						**Máquina 5**						**Máquina 6**					
Jobs	***ijk***	r_i	p_i	d_i	w_i	***Jobs***	***ijk***	r_i	p_i	d_i	w_i	***Jobs***	***ijk***	r_i	p_i	d_i	w_i
1	144	10	7	38	1	1	156	20	6	47	1	1	165	17	3	41	1
2	246	43	4	47	1	2	253	13	10	23	1	2	264	23	10	33	1
3	342	5	4	22	1	3	356	27	7	47	1	3	363	9	8	30	1
4	444	15	3	30	1	4	455	18	8	38	1	4	466	26	9	47	1
5	546	24	1	47	1	5	553	12	5	39	1	5	564	17	4	43	1
6	642	3	3	23	1	6	655	25	4	46	1	6	663	6	9	32	1

3º Passo: Aplicação do AG a cada um dos problemas de uma só máquina

Aqui, aplicamos o AG a cada um dos problemas de uma só máquina obtidos no passo anterior, com o objectivo de encontrar a solução óptima ou quase-óptima para cada um desses problemas. Lembramos que, o critério de desempenho que se pretende optimizar neste exemplo é a minimização do tempo total de finalização da produção $C_{\max}$ (*makespan*). Assim, após a aplicação do AG a cada um dos problemas de uma só máquina, obtiveram-se as seguintes soluções:

Máquina 1: S_{opt}=(1,4,6,3,2,5)

Máquina 2: S_{opt}=(2,4,6,5,1,3)

Máquina 3: S_{opt}=(3,1,2,5,4,6)

Máquina 4: S_{opt}=(6,4,3,1,5,2)

Máquina 5: S_{opt}=(2,4,5,6,3,1)

Máquina 6: S_{opt}=(6,3,2,1,5,4)

De seguida, as sequências de *jobs* encontradas pelo AG são convertidas nas respectivas sequências das operações *ijk*. Deste modo, para as sequências de *jobs* apresentadas acima as respectivas sequências das operações *ijk* são:

Máquina 1: S_{opt}=(112, 412, 614, 314, 215, 515)

Máquina 2: S_{opt}=(221, 421, 621, 522, 123, 325)

Máquina 3: S_{opt}=(331, 131, 232, 531, 433, 636)

Máquina 4: S_{opt}=(642, 444, 342, 144, 546, 246)

Máquina 5: S_{opt}=(253, 455, 553, 655, 356, 156)

Máquina 6: S_{opt}=(663, 363, 264, 165, 564, 466)

4º Passo: Integração das soluções obtidas no problema principal

A tabela 7.13 apresenta os tempos início T_i e os tempos de finalização T_f de cada operação nas 6 máquinas. Com base nos dados da tabela 7.13 obtém-se o sequenciamento apresentado na figura 7.21.

Tabela 7.13 Tempos de início e de finalização das operações dos *jobs* nas máquinas

Máquina 1			**Máquina 2**			**Máquina 3**			**Máquina 4**			**Máquina 5**			**Máquina 6**		
ijk	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f
112	1	4	221	0	8	331	0	5	642	3	6	253	13	23	663	6	15
412	5	10	421	8	13	131	5	6	444	15	18	455	23	31	363	15	23
614	15	25	621	13	16	232	8	13	342	18	22	553	31	36	264	23	33
314	25	34	522	16	19	531	13	22	144	22	29	655	36	40	165	33	36
215	34	44	123	19	25	433	22	27	546	29	30	356	40	47	564	36	40
515	44	47	325	26	27	636	29	30	246	43	47	156	47	53	466	40	49

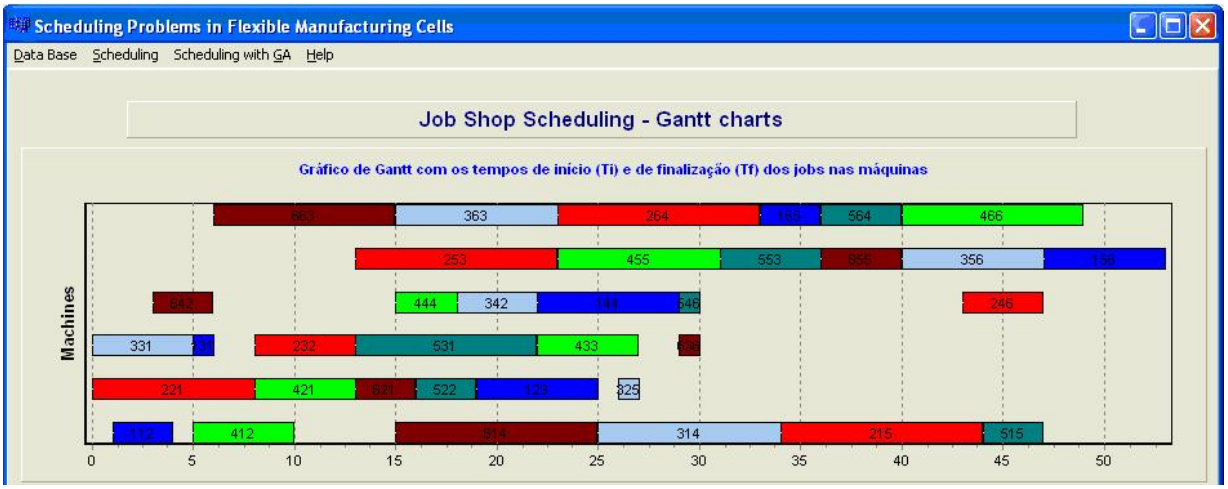


Figura 7.21 Sequenciamento resultante da integração das soluções obtidas no 3º passo.

5º Passo: Verificação da exequibilidade da solução

O sequenciamento apresentado na figura 7.21 não é exequível pelo facto de haver várias operações sobrepostas, como por exemplo: a operação O_{412} com a operação O_{421}. Na tabela 7.13, os tempos de início (T_i) e os tempos de finalização (T_f) de cada operação foram determinados com base, unicamente, nos instantes de chegada (r_i), tempos de processamento (p_i) e respectivas sequências das operações nas máquinas. Como já foi referido, nos exemplos anteriores, o algoritmo até ao 4º passo não entra em linha de conta com a precedência imposta pela ocupação das máquinas.

Uma vez que o sequenciamento obtido na figura 7.21 não é exequível, é necessário proceder à coordenação das operações dos *jobs* nas várias máquinas. A tabela 7.14 apresenta os "novos" tempos de início T'_i e de finalização T'_f de cada operação, com o objectivo de tornar exequível o sequenciamento apresentado na figura 7.21. Estes valores resultam da aplicação do mecanismo de coordenação das operações (MCO) dos vários *jobs* nas máquinas. O segundo gráfico de *Gantt* apresentado na figura 7.22 mostra o sequenciamento exequível, resultante dos tempos de início e de finalização (T'_i e T'_f), estes, apresentados na tabela 7.14.

Tabela 7.14 Novos tempos de início e de finalização das operações dos *jobs* nas máquinas

Máquina 1			**Máquina 2**			**Máquina 3**			**Máquina 4**			**Máquina 5**			**Máquina 6**		
ijk	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f	***ijk***	T_i	T_f
112	6	9	221	0	8	331	0	5	342	5	9	253	13	23	363	9	17
412	13	18	421	8	13	131	5	6	642	16	19	553	25	30	663	19	28
314	18	27	621	13	16	232	8	13	144	22	29	356	30	37	264	28	38
614	28	38	123	16	22	531	13	22	444	29	32	455	37	45	165	38	41
215	38	48	522	22	25	433	22	27	246	48	52	655	45	49	564	41	45
515	48	51	325	27	28	636	49	50	546	52	53	156	49	55	466	45	54

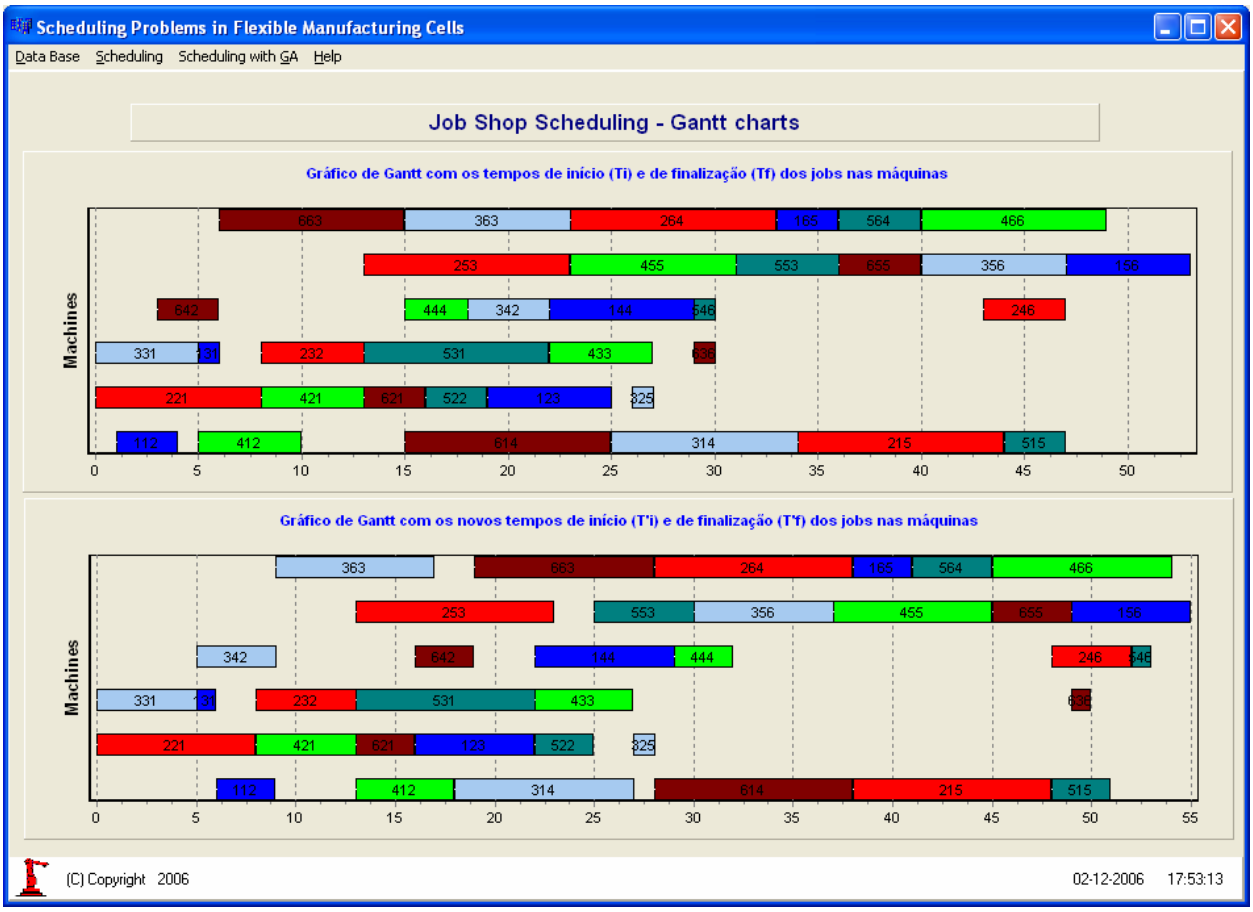


Figura 7.22 Sequenciamento resultante da aplicação do 5º passo do algoritmo.

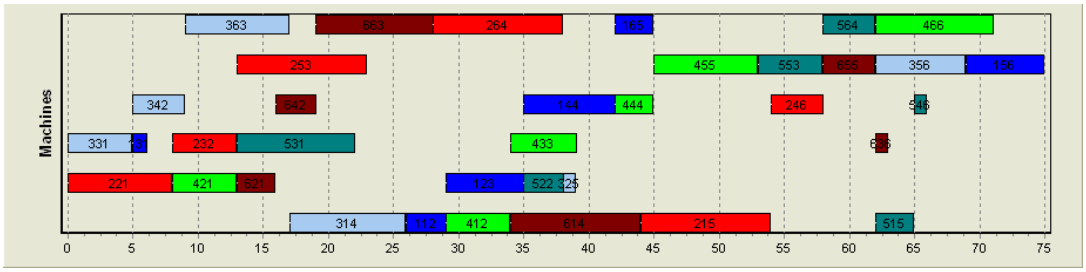
Vamos utilizar este exemplo para explicarmos melhor o princípio de funcionamento do mecanismo de coordenação das operações (MCO), anteriormente abordado na subsecção 7.5.2.1. A figura 7.23(a) apresenta o sequenciamento resultante da não aplicação do MCO, onde o *makespan* é 75. Neste sequenciamento todas as operações foram sequenciadas para a frente, resultando, desta forma, espaços livres consideráveis entre operações. Percebemos facilmente que este sequenciamento fica muito aquém de uma boa solução. O sequenciamento apresentado na figura 7.23(b) foi obtido com o recurso à função *PreenchimentoEspacosLivresParaTras--SemArrastamento()*, isto é, sempre que possível, as operações preencheram os espaços livres para trás sem causarem arrastamentos nas operações a jusante. Como exemplo, referimos a operação O_{112} que sofreu um sequenciamento para trás sem causar qualquer arrastamento nas operações a jusante. O sequenciamento resultante da aplicação da função *PreenchimentoEspacosLivresParaTrasSemArrastamento()* apresenta um *makespan* de 57, tendo este melhorado significativamente comparativamente à solução da figura 7.23(a). Assim, facilmente percebemos que o preenchimento de espaços livres para trás sem arrastamento contribuem sempre para uma melhoria da solução final. Desta forma, os sequenciamentos apresentados em (c), (d), (e) e (f) da figura 7.23 foram obtidos recorrendo-se a esta função.

O preenchimento de espaços livres para trás com arrastamento é efectuado pela função *PreenchimentoEspacosLivresParaTrasComArrastamento()*. Este tipo de preenchimento é feito sempre que o espaço livre encontrado seja inferior ao tempo de processamento da operação p_{ijk} a sequenciar. Deste tipo de sequenciamento resulta sempre um arrastamento das operações a jusante do espaço livre ocupado. No entanto, este tipo de sequenciamento pode contribuir para uma melhoria da solução. Vejamos o sequenciamento apresentado na figura 7.23(c) onde foi aplicada a função *PreenchimentoEspacosLivresParaTrasComArrastamento()* unicamente à máquina 1. A operação O_{412} preencheu um espaço livre para trás provocando um arrastamento na operação a jusante O_{314}. Mas, apesar do *makespan* não ter melhorado relativamente à situação da figura 7.23(b) (*makespan*=57), verificamos uma melhoria nas sequências das máquinas 3 e 4. Sempre que ocorre uma situação destas o preenchimento do espaço livre deve ser considerado. No entanto, nem todos os preenchimentos de espaços livres para trás com arrastamento contribuem para uma melhoria das sequências nas máquinas e, consequentemente, para uma melhoria da solução final. Exemplo disso é o sequenciamento apresentado na figura 7.23(d) onde a operação O_{531} ao preencher um espaço livre para trás originou um *makespan* de 62. Sempre que ocorre uma situação destas, o preenchimento do espaço livre nessa máquina não é considerado.

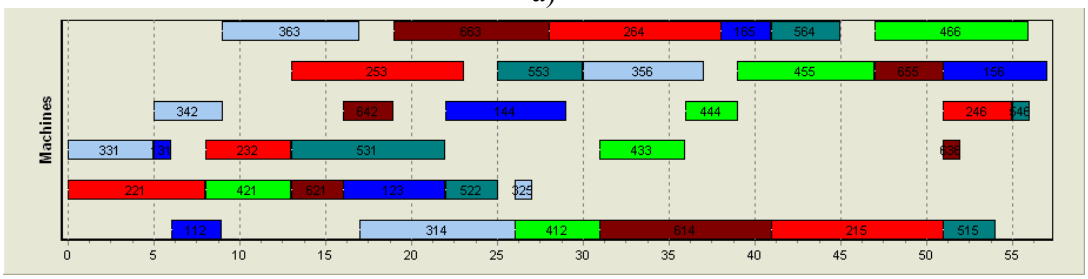
O sequenciamento apresentado na figura 7.23(e) mostra a solução óptima da instância FT06, *makespan* igual a 55. Se compararmos o diagrama de *Gantt* da figura 7.23(c) com o da figura 7.23(e), podemos concluir que foi o sequenciamento para trás da operação O_{356} que contribuiu, também, para a obtenção da solução óptima.

O diagrama de *Gantt* apresentado na figura 7.23(f) resultou da aplicação das duas funções, acima referidas, a todas as máquinas. Como podemos verificar nesta figura, o sequenciamento para trás com arrastamento da operação O_{531} prejudicou significativamente o resultado final, sendo neste caso o *makespan* de 60.

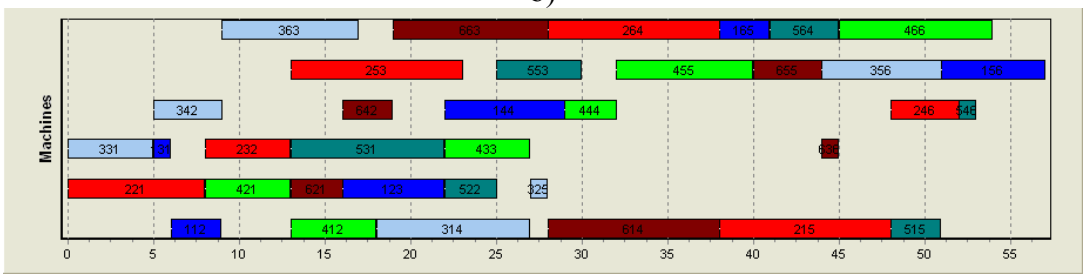
Assim, e para concluirmos, a solução óptima foi obtida através da aplicação da função *PreenchimentoEspacosLivresParaTrasSemArrastamento()* a todas as máquinas e da aplicação da função *PreenchimentoEspacosLivresParaTrasComArrastamento()* unicamente às máquinas 1 e 5.



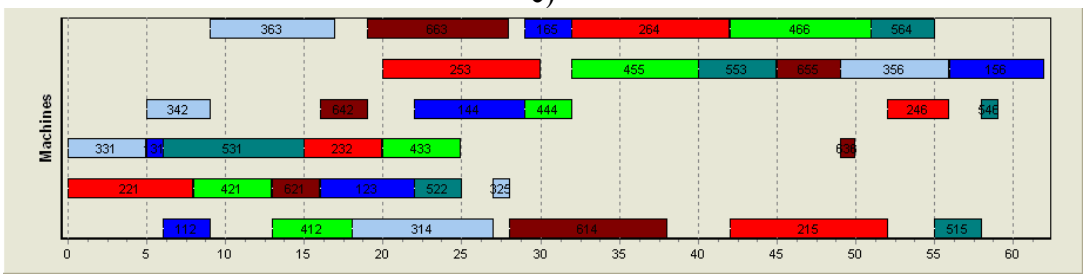
a)



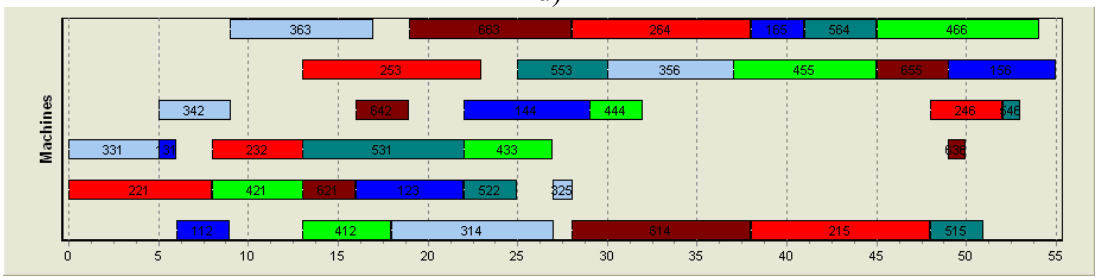
b)



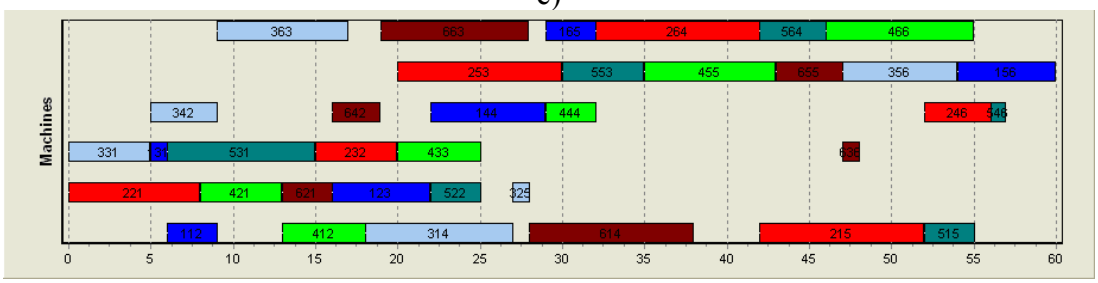
c)



d)



e)



f)

Figura 7.23 Sequenciamentos possíveis para o problema FT06.

7.7 Testes e resultados computacionais

Nesta secção apresentamos um conjunto de testes e resultados computacionais obtidos pelo HybFlexGA e por diversos algoritmos desenvolvidos por vários autores. Nos testes computacionais são utilizadas instâncias de problemas de sequenciamento do tipo *Job-Shop* retiradas da OR-Library [http#4] (ver o apêndice B). Os referidos testes computacionais têm como finalidade a comparação da performance do HybFlexGA com outros algoritmos desenvolvidos por diferentes autores. Todos os testes computacionais realizados no HybFlexGA foram efectuados num computador *Pentium* IV, com 3GHz e 1 GB de RAM.

7.7.1 Resultados computacionais obtidos nas instâncias Fisher e Thompson

Com o objectivo de testar o desempenho do HybFlexGA foram seleccionadas e examinadas algumas instâncias de referência – *benchmark*. As instâncias FT06, FT10 e FT20 apresentadas por Fisher e Thompson em [Fisher e Thompson, 1963], são as mais conhecidas na comunidade científica. A segunda instância FT10 (10 *jobs* e 10 máquinas) tem sido utilizada por muitos investigadores em testes computacionais. Carlier e Pinson em 1989 obtiveram para esta instância uma solução com o *makespan* de 930 e demonstraram que a referida solução é a solução óptima do problema [Carlier e Pinson, 1989]. As soluções óptimas para as outras *benchmarks* são 55 e 1165, respectivamente.

Nestes testes computacionais, os parâmetros do algoritmo genético utilizado no HybFlexGA foram:

- população inicial (Ψ_t): gerada aleatoriamente;
- operador de cruzamento: *two-point crossover: 4 children* (TPC4C);
- probabilidade de cruzamento (P_c): 1,0;
- operador de mutação: *arbitrary 20%-job change* (Arb20%JC);
- probabilidade de mutação (P_m): 1,0;
- tamanho da população (N_{pop}): 40 cromossomas;
- condição de paragem: 10 gerações para a instância FT06 e 30 gerações para as instâncias FT10 e FT20.

Os resultados computacionais obtidos com o HybFlexGA foram comparados com os resultados obtidos por Wu, Xing, Lee, Zhou e Liang em [Wu *et al.*, 2004] e, ainda, com os resultados obtidos pelo software LEKIN, utilizando a rotina *General Shifting Bottleneck (SB) Routine* [http#5]. A heurística SB é uma das mais bem sucedidas na minimização do *makespan* em

problemas de sequenciamento do tipo *Job-Shop*. Como podemos verificar na tabela 7.15, o HybFlexGA obteve bons resultados nas instâncias FT06, FT10 e FT20. Dos resultados apresentados na tabela 7.15 podemos retirar as seguintes conclusões:

- na instância FT06, o HybFlexGA e o algoritmo utilizado em [Wu *et al.*, 2004] obtiveram a solução óptima nos cinco testes realizados, no entanto, o HybFlexGA foi o mais rápido. A rotina *general SB routine* [http#5], apesar de ter gasto menos tempo (1 segundo), não obteve a solução óptima em nenhum dos testes realizados;
- relativamente à instância FT10, a rotina *general SB routine* [http#5] foi a que necessitou de menos tempo de computação (2 segundos), mas também foi a que obteve piores resultados de *makespan* e, consequentemente, o maior erro relativo percentual médio (17,63%). O HybFlexGA foi o único que, nos cinco testes realizados, encontrou uma solução óptima (teste 3), o que obteve melhor média de *makespan* (942) e o que conseguiu o melhor erro percentual médio (1,23%);
- em relação à instância FT20, o HybFlexGA foi o que obteve melhores resultados do ponto de vista de tempo médio de computação (10,4 segundos), o que conseguiu a melhor média de *makespan* (1183) e, por conseguinte, o melhor erro relativo percentual médio (1,53%).

Tabela 7.15 Resultados computacionais obtidos pelo HybFlexGA, [Wu *et al.*, 2004] e *General SB Routine* [http#5]

		HybFlexGA			**Resultados em [Wu *et al.*, 2004]**			**General SB Routine [http#5]**		
Inst.	**Testes**	**Tempo (seg.)**	***Makespan***	**Erro (%)**	**Tempo (seg.)**	***Makespan***	**Erro (%)**	**Tempo (seg.)**	***Makespan***	**Erro (%)**
FT06 (55)	**1**	2	55	0	2,13	55	0	1	59	7,27
	2	2	55	0	2,35	55	0	1	59	7,27
	3	2	55	0	2,34	55	0	1	59	7,27
	4	2	55	0	2,47	55	0	1	59	7,27
	5	2	55	0	2,72	55	0	1	59	7,27
	Média	**2**	**55**	**0**	**2,40**	**55**	**0**	**1**	**59**	**7,27**
FT10 (930)	**1**	9	944	1,51	102,98	1028	10,54	2	1094	17,63
	2	8	948	1,94	92,30	998	7,31	2	1094	17,63
	3	9	930	0	86,53	1043	12,15	2	1094	17,63
	4	9	949	2,04	101,88	1017	9,35	2	1094	17,63
	5	8	937	0,75	95	998	7,31	2	1094	17,63
	Média	**8,6**	**942**	**1,23**	**95,74**	**1017**	**9,33**	**2**	**1094**	**17,63**
FT20 (1165)	**1**	10	1180	1,23	57,54	1236	6,09	16	1291	10,82
	2	11	1171	0,52	60,75	1265	8,58	15	1291	10,82
	3	10	1169	0,34	49,88	1247	7,04	15	1291	10,82
	4	10	1204	3,35	72,59	1241	6,52	16	1291	10,82
	5	11	1191	2,23	42,62	1263	8,41	16	1291	10,82
	Média	**10,4**	**1183**	**1,53**	**56,68**	**1250**	**7,33**	**15,6**	**1291**	**10,82**

O erro relativo percentual da tabela 7.15 foi obtido através da seguinte expressão:

$$erro-relativo(\%) = \frac{Valor\ \ encontrado - Valor\ \ óptimo}{Valor\ \ óptimo} \times 100\% \tag{7.6}$$

Fazendo agora uma análise global aos resultados computacionais obtidos na tabela 7.15, podemos concluir que:

- o HybFlexGA foi o que obteve melhor desempenho nas três instâncias, comparativamente aos resultados obtidos por [Wu *et al.*, 2004] e pela rotina *general SB routine*;
- o algoritmo utilizado por [Wu *et al.*, 2004] foi o que necessitou de maior tempo de computação nas três instâncias;
- a rotina *general SB routine* obteve nas três instâncias os piores resultados de *makespan* e, por conseguinte, os maiores erros relativos percentuais.

Relativamente às três instâncias Fisher e Thompson, outros resultados computacionais recentes foram publicados por Yang em [Yang e Wang, 2000], [Yang, 2005] e [Yang, 2006]. No seu trabalho mais recente [Yang, 2006], Yang apresenta dois modelos de uma rede neuronal adaptativa a que chama *Constraint Satisfaction Adaptive Neural Network* (CSANN-II e CSANN--LS), para a resolução de problemas de sequenciamento do tipo *Job-Shop*. O autor desenvolveu todo o seu trabalho utilizando a linguagem de programação C++ e usou nos seus testes computacionais um computador *Pentium* IV de 2.8 GHz. O referido autor efectuou 50 testes computacionais para cada uma das instâncias de Fisher e Thompson e comparou os valores mínimos, médias, desvios padrões e tempos de processamento obtidos pelos CSANN-II e CSANN-LS com os algoritmos GT-Random e GT-Rule. Uma vez que também utilizamos no desenvolvimento do HybFlexGA a linguagem de programação C++ e nos testes computacionais um computadores com características muito semelhantes ao utilizado por Yang, resolvemos comparar os testes computacionais obtidos pelo HybFlexGA na tabela 7.15 com os testes computacionais obtidos por Yang em [Yang, 2006]. Na tabela 7.16 encontram-se os resultados computacionais obtidos pelo HybFlexGA e os resultados obtidos por Yang [Yang, 2006], onde *Men/Med/DesPad* significa menor valor, média e desvio padrão.

Os resultados computacionais apresentados nas tabelas 7.15 e 7.16 mostram que o HybFlexGA apresenta um bom desempenho na resolução de problemas de sequenciamento do tipo *Job-Shop*. Os tempos computacionais apresentados na tabela 7.16 mostram que os modelos com base em redes neuronais desenvolvidos por Yang têm tempos computacionais significativamente maiores,

quando comparados com os tempos obtidos pelo HybFlexGA. Podemos concluir, com base nos testes computacionais obtidos nas instâncias FT06, FT10 e FT20, que o algoritmo de sequenciamento implementado no HybFlexGA pode ser utilizado na obtenção de soluções óptimas (ou quase óptimas) de problemas de sequenciamento do tipo *Job-Shop*. Assim, o HybFlexGA pode ser muito útil na resolução deste tipo de problemas.

Tabela 7.16 Testes computacionais para comparação de algoritmos

	Algoritmos	**FT06**	**FT10**	**FT20**
Makespan (Men/Med/DesPad)	HybFlexGA	55/55/0	930/942/7,2	1169/1183/13,1
	CSANN-LS	55/55/0	971/999/16,1	1221/1269/25,8
	CSANN-II	55/55/0	982/1009/9,9	1292/1334/12,7
	GT-Random	55/56,2/0,8	1048/1102/17,9	1336/1383/16,7
	GT-Rule	55/56,8/0,8	1073/1116/15,7	1333/1379/17,4
Tempo (seg.) (Men/Med/DesPad)	HybFlexGA	2/2/0	8/8,6/0,5	10/10,4/0,5
	CSANN-LS	16/78,3/105,5	68/675/916,2	49/100/61,5
	CSANN-II	31/129/166,5	222/753/607,1	820/941/168,6
	GT-Random	4/4,3/0,46	16/16,9/0,56	33/34,2/1,0
	GT-Rule	4/4,6/0,49	17/18/0,45	35/38,9/3,9

Dada a popularidade das instâncias de Fisher e Thompson decidimos apresentar a seguir as respectivas sequências óptimas obtidas pelo HybFlexGA.

Sequência óptima obtida pelo HybFlexGA para a instância FT06:

Máquina 1: $S_{óptima}=\{J_1, J_4, J_3, J_6, J_2, J_5\}$

Máquina 2: $S_{óptima}=\{J_2, J_4, J_6, J_1, J_5, J_3\}$

Máquina 3: $S_{óptima}=\{J_3, J_1, J_2, J_5, J_4, J_6\}$

Máquina 4: $S_{óptima}=\{J_3, J_6, J_1, J_4, J_2, J_5\}$

Máquina 5: $S_{óptima}=\{J_2, J_5, J_3, J_4, J_6, J_1\}$

Máquina 6: $S_{óptima}=\{J_3, J_6, J_2, J_1, J_5, J_4\}$

Sequência óptima obtida pelo HybFlexGA para a instância FT10:

Máquina 1: $S_{óptima}=\{J_9, J_1, J_2, J_7, J_4, J_5, J_{10}, J_8, J_6, J_3\}$

Máquina 2: $S_{óptima}=\{J_4, J_6, J_7, J_{10}, J_9, J_5, J_3, J_8, J_1, J_2\}$

Máquina 3: $S_{óptima}=\{J_6, J_4, J_5, J_8, J_2, J_{10}, J_7, J_9, J_1, J_3\}$

Máquina 4: $S_{óptima}=\{J_6, J_7, J_9, J_5, J_3, J_1, J_2, J_{10}, J_4, J_8\}$

Máquina 5: $S_{óptima}=\{J_4, J_2, J_5, J_6, J_8, J_1, J_9, J_{10}, J_7, J_3\}$

Máquina 6: $S_{óptima}=\{J_6, J_5, J_9, J_7, J_8, J_{10}, J_1, J_3, J_2, J_4\}$

Máquina 7: $S_{óptima}=\{J_4, J_{10}, J_7, J_6, J_9, J_8, J_1, J_2, J_3, J_5\}$

Máquina 8: $S_{óptima}=\{J_4, J_6, J_5, J_9, J_7, J_3, J_1, J_8, J_2, J_{10}\}$

Máquina 9: $S_{óptima}=\{J_6, J_4, J_{10}, J_5, J_7, J_3, J_8, J_9, J_1, J_2\}$

Máquina 10: $S_{óptima}=\{J_6, J_2, J_7, J_9, J_{10}, J_5, J_8, J_4, J_3, J_1\}$

Sequência óptima obtida pelo HybFlexGA para a instância FT20:

Máquina 1: $S_{óptima}=\{J_{17}, J_5, J_{20}, J_{15}, J_9, J_{13}, J_1, J_2, J_8, J_{16}, J_{11}, J_{18}, J_{12}, J_{19}, J_{10}, J_3, J_{14}, J_6, J_7, J_4\}$

Máquina 2: $S_{óptima}=\{J_{16}, J_5, J_{17}, J_1, J_{20}, J_{15}, J_6, J_8, J_{13}, J_1, J_{10}, J_{19}, J_3, J_2, J_{18}, J_{12}, J_7, J_4, J_{14}, J_9\}$

Máquina 3: $S_{óptima}=\{J_5, J_{17}, J_6, J_8, J_{20}, J_{13}, J_{12}, J_1, J_{10}, J_{19}, J_{14}, J_{15}, J_3, J_9, J_7, J_2, J_{16}, J_{18}, J_4, J_{11}\}$

Máquina 4: $S_{óptima}=\{J_5, J_{17}, J_9, J_{20}, J_{11}, J_{13}, J_8, J_{16}, J_1, J_{19}, J_2, J_{15}, J_{10}, J_{12}, J_3, J_7, J_{14}, J_{18}, J_4, J_6\}$

Máquina 5: $S_{óptima}=\{J_5, J_{17}, J_6, J_{20}, J_{15}, J_{11}, J_{13}, J_8, J_{16}, J_1, J_{19}, J_{18}, J_3, J_{10}, J_{12}, J_4, J_2, J_9, J_{14}, J_7\}$

7.7.2 Comparação dos resultados obtidos pelo HybFlexGA com outros resultados

A finalidade desta subsecção é comparar os resultados obtidos pelo HybFlexGA com os resultados obtidos por outros investigadores. Muitas vezes, os trabalhos científicos publicados não referem com clareza as condições em que são efectuados os testes computacionais, como por exemplo: as características do computador utilizado nos testes, o número de testes realizados, os tempos de computação obtidos, entre outras. Assim, por vezes, não é fácil compararmos os resultados obtidos com os resultados obtidos por outros investigadores.

A tabela 7.17 apresenta os melhores valores obtidos por vários algoritmos encontrados na literatura para as instâncias de Fisher e Thompson (FT) e para algumas instâncias de Lawrence (LA). Estas instâncias estão disponíveis em muita da bibliografia apresentada, mas podem ser facilmente obtidas pela *Internet* [http#4]. O apêndice B apresenta um conjunto de tabelas com as instâncias FT e LA utilizadas nos testes computacionais. Para cada uma destas instâncias efectuamos o levantamento dos melhores valores encontrados pelos vários algoritmos, utilizados com mais frequência pela comunidade científica. Assim, na tabela 7.17 encontram-se os melhores valores obtidos pelos seguintes algoritmos: PLGA na coluna (5) [Fa e Lin, 2003], *Genetic Algorithm* (GA) na coluna (6) [Wang e Zheng, 2001], *Simulated Annealing* (SA) na coluna (7) [Van *et al.*, 1992], *Tabu Search* (TS) na coluna (8) [Amico *et al.*, 1993], *Hybrid Particle Swarm Optimization* (HPSO) na coluna (9) [Xia *et al.*, 2004], *Shifting Bottleneck* (SB) na coluna (10) [Adams *et al.*, 1988], *Modified Genetic Algorithm* (MGA) na coluna (11) [Wang e Zheng, 2002], *Simulated Annealing* (SA) na coluna (12) [Wang e Zheng, 2002] e *Hybrid Genetic Algorithm* (HGA) na coluna (13) [Lui e Xi, 2006]. Os autores Hong e Jian em [Hong e Jian, 2006] também publicaram os resultados que se encontram nas colunas (7), (8) e (10), relativamente às instâncias LA.

Como já foi referido, os melhores valores encontrados por alguns dos algoritmos mais utilizados pela comunidade científica encontram-se na tabela 7.17, estando os valores óptimos a negrito. Por exemplo, para cada instância, a coluna (5) apresenta o melhor valor obtido pelo algoritmo PLGA [Fa e Lin, 2003] em 30 testes computacionais e a coluna (13) mostra também o melhor valor obtido pelo HGA [Lui e Xi, 2006] em 20 testes computacionais. Nas restantes colunas, o número de testes realizados variaram de coluna para coluna, no entanto, os autores dos respectivos trabalhos apresentaram sempre a melhor solução encontrada para cada instância nos testes computacionais. Com o objectivo de permitir a comparação dos resultados obtidos pelo HybFlexGA com os resultados obtidos por outros autores, efectuámos 10 testes computacionais para cada instância e registámos na tabela 7.17 o melhor valor encontrado. Nestes testes computacionais, os parâmetros do algoritmo genético utilizado no HybFlexGA foram:

– população inicial (Ψ_t): gerada aleatoriamente;

– operador de cruzamento: *two-point crossover: 4 children* (TPC4C);

– probabilidade de cruzamento (P_c): 1,0;

– operador de mutação: *arbitrary 20%-job change* (Arb20%JC);

– probabilidade de mutação (P_m): 1,0;

– tamanho da população (N_{pop}): 40 cromossomas para as instâncias FT06, FT10 e FT20; 60 cromossomas para as instâncias de Lawrence (LA);

– condição de paragem: 30 gerações para as instâncias FT06, FT10 e FT20; 100 gerações para todas as instâncias de Lawrence (LA).

Tabela 7.17 Resultados obtidos pelo HybFlexGA e por outros algoritmos de vários autores

(1) Problema	(2) *nxm*	(3) Valor óptimo	(4) HybFlexGA	(5) PLGA	(6) GA	(7) SA	(8) TS	(9) HPSO	(10) SB	(11) MGA	(12) SA	(13) HGA
FT06	6x6	55	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**
FT10	10x10	930	**930**	**930**	**930**	**930**	935	**930**	**930**	**930**	939	936
FT20	20x5	1165	1169	1173	**1165**	**1165**	**1165**	1178	1178	**1165**	1227	1175
LA01	10x5	666	**666**	**666**	**666**	**666**	**666**	**666**	**666**	**666**	**666**	**666**
LA06	15x5	926	**926**	**926**	**926**	**926**	**926**	**926**	**926**	**926**	**926**	**926**
LA11	20x5	1222	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**
LA16	10x10	945	**945**	**945**	**945**	956	**945**	**945**	978	**945**	979	**945**
LA21	15x10	1046	1052	1051	1058	1063	1048	1047	1048	1058	1083	1067
LA26	20x10	1218	**1218**	**1218**	**1218**	**1218**	**1218**	**1218**	1224	**1218**	1253	**1218**
LA31	30x10	1784	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**
LA36	15x15	1268	1278	1279	1292	1293	1278	1269	1305	1291	1321	1292
Número de soluções óptimas			**8**	**8**	**9**	**8**	**8**	**8**	**6**	**9**	**5**	**7**

Após uma leitura dos resultados apresentados na tabela 7.17, verificamos que, de uma forma geral, o HybFlexGA apresenta uma boa performance relativamente aos restantes algoritmos em análise. Apesar de termos realizado 10 testes computacionais por instância, o HybFlexGA encontrou a solução óptima em 8 instâncias das 11 analisadas. O HybFlexGA não encontrou a

solução óptima nas instâncias FT20, LA21 e LA36, no entanto, os valores obtidos pelo HybFlexGA nessas instâncias encontram-se muito perto dos respectivos valores óptimos. Assim, os resultados obtidos nos 10 testes computacionais pelo HybFlexGA levam-nos a concluir que este é eficaz na resolução de problemas de sequenciamento do tipo *Job-Shop*.

Face aos resultados obtidos pelo HybFlexGA nos 10 testes computacionais (coluna 4 da tabela 7.17), ponderámos a possibilidade de realizarmos um maior número de testes computacionais em cada instância, uma vez que o aumento do número de testes computacionais poderia aumentar a possibilidade do HybFlexGA obter melhores soluções. Assim, realizamos 30 testes computacionais para cada uma das instâncias da tabela 7.17. Nos 30 testes computacionais realizados, o HybFlexGA encontrou a solução óptima na instância FT20 (1165), mas nas restantes instâncias o HybFlexGA encontrou os mesmos resultados obtidos nos 10 computacionais. Desta forma, podemos concluir que, o aumento do custo computacional no HybFlexGA (*e.g.*, aumento do número de testes, aumento do número de gerações no AG, entre outros) pode não contribuir de forma significativa para uma melhoria da solução final.

Considerando a solução óptima obtida pelo HybFlexGA na instância FT20 durante a realização dos 30 testes computacionais, podemos concluir que:

- o HybFexGA na coluna (4), o GA na coluna (6) [Wang e Zheng, 2001] e o MGA na coluna (11) [Wang e Zheng, 2002] são os que apresentam melhor desempenho, tendo obtido nove instâncias com soluções óptimas (FT06, FT10, FT20, LA01, LA06, LA11, LA16, LA26 e LA31). Nas instâncias onde não foi obtida a solução óptima (LA21 e LA36), verificamos que o HybFlexGA foi o que obteve valores mais próximos dos valores óptimos. Os resultados obtidos pelo HybFlexGA são reveladores da sua eficiência na resolução de problemas de sequenciamento do tipo *Job-Shop*;
- o PLGA na coluna (5) [Fa e Lin, 2003], o SA na coluna (7) [Van *et al.*, 1992], o TS na coluna (8) e [Amico *et al.*, 1993] e o HPSO na coluna (9) [Xia *et al.*, 2004] obtiveram oito instâncias com soluções óptimas;
- o SA na coluna (12) [Wang e Zheng, 2002], SB na coluna (10) [Adams *et al.*, 1988] e o HGA na coluna (13) [Lui e Xi, 2006] obtiveram os piores resultados, tendo cada um destes algoritmos obtido cinco, seis e sete instâncias com soluções óptimas respectivamente;
- nenhum dos algoritmos utilizados na tabela 7.17 encontrou a solução óptima para as instâncias LA21 e LA36, em virtude destas instâncias serem de difícil resolução.

Em suma, os resultados obtidos na tabela 7.17 atestam a eficiência do HybFlexGA na resolução de problemas de sequenciamento do tipo *Job-Shop*.

7.7.3 Comparação do HybFlexGA com o LEKIN

Esta subsecção apresenta um conjunto de testes computacionais realizados com o objectivo de compararmos os softwares HybFlexGA e LEKIN. Como foi referido na secção 7.3, o LEKIN foi desenvolvido por uma vasta equipa de investigadores, sob a orientação e supervisão de Michael Pinedo [Pinedo, 1995], [Pinedo, 2002]. De referir que, a versão 2.4 deste software encontra-se disponível na *Internet* para *download* [http#5]. O LEKIN tem incorporado um conjunto de algoritmos de sequenciamento, tais como: *Earliest Due Date* (EDD), *Shortest Processing Time* (STP), *General Shifting Bottleneck (SB) Routine*, *Local Search,* entre outros. Como já foi referido anteriormente, a heurística SB é uma das mais bem sucedidas na minimização do *makespan* em problemas de sequenciamento do tipo *Job-Shop*.

Com o objectivo de tornar a comparação entre o HybFlexGA e LEKIN justa, os testes computacionais foram realizados no mesmo computador (*Pentium* IV, 3GHz e 1 GB de RAM). Em cada instância, foram realizados 10 testes computacionais no HybFlexGA e para cada um dos algoritmos de sequenciamento disponibilizados pelo LEKIN. A tabela 7.18 apresenta as várias médias obtidas nos testes computacionais, para as três instâncias FT e para as oito instâncias LA. Assim, na coluna (5) estão registadas as médias dos tempos de sequenciamento em segundos $\overline{CPU}$, na coluna (6) as médias do tempo máximo de conclusão $\overline{C}_{\max}$, na coluna (7) as médias do máximo atraso $\overline{T}_{\max}$, na coluna (8) as médias do somatório dos *jobs* em atraso $\overline{\sum U_i}$ e na coluna (9) as médias do somatório dos tempos máximos de conclusão $\overline{\sum C_i}$. Todas as médias apresentadas nesta tabela foram arredondadas às unidades.

Os resultados computacionais apresentados na tabela 7.18 mostram que:

- em relação à coluna (5), as regras de sequenciamento EDD e SPT são pouco exigentes do ponto de vista computacional, tendo obtido, em todos os testes realizados, tempos de computação médios da ordem de 1 segundo. Os restantes algoritmos apresentam médias dos tempos de sequenciamento $\overline{CPU}$ variáveis, estando estas directamente relacionadas com a complexidade da instância. Podemos afirmar que o HybFlexGA apresenta tempos de computação satisfatórios comparativamente aos restantes algoritmos utilizados nos testes computacionais;

Tabela 7.18 Resultados computacionais obtidos pelo HybFlexGA e LEKIN

(1) Instância	(2) *n* x *m*	(3) Valor óptimo	(4) Sequenciamento		(5) $\overline{CPU}$ (seg.)	(6) $\overline{C}_{\max}$	(7) $\overline{T}_{\max}$	(8) $\overline{\sum U_i}$	(9) $\overline{\sum C_i}$
FT06	6x6	55	HybFlexGA		2	55	0	0	291
			LEKIN	EDD	1	63	8	2	321
				SPT	1	73	18	4	336
				General SB Routine	1	59	4	1	296
				Local Search	3	55	0	0	291
FT10	10x10	930	HybFlexGA		9	942	12	1	8035
			LEKIN	EDD	1	1246	316	10	10048
				SPT	1	1338	408	10	10713
				General SB Routine	2	1094	164	10	9951
				Local Search	61	944	14	2	8052
FT20	20x5	1165	HybFlexGA		11	1183	18	2	5762
			LEKIN	EDD	1	1672	507	12	6309
				SPT	1	1558	393	4	6794
				General SB Routine	15	1291	126	9	6345
				Local Search	26	1165	0	0	5467
LA01	10x5	666	HybFlexGA		4	666	0	0	3157
			LEKIN	EDD	1	865	199	3	3546
				SPT	1	1122	456	4	4770
				General SB Routine	2	686	20	3	3287
				Local Search	5	666	0	0	3249
LA06	15x5	926	HybFlexGA		10	926	0	0	4445
			LEKIN	EDD	1	1024	98	3	4672
				SPT	1	1475	549	4	6478
				General SB Routine	5	926	0	0	4438
				Local Search	6	926	0	0	4492
LA11	20x5	1222	HybFlexGA		10	1229	7	1	5879
			LEKIN	EDD	1	1272	50	1	5884
				SPT	1	1802	580	4	8299
				General SB Routine	25	1235	13	2	5937
				Local Search	6	1222	0	0	5876
LA16	10x10	945	HybFlexGA		12	1033	88	2	9266
			LEKIN	EDD	1	1143	198	2	9372
				SPT	1	1532	587	3	11257
				General SB Routine	5	1106	161	7	10376
				Local Search	60	949	4	1	8683
LA21	15x10	1046	HybFlexGA		14	1175	129	4	11147
			LEKIN	EDD	1	1440	394	9	12684
				SPT	1	1502	456	5	13067
				General SB Routine	18	1211	165	10	11669
				Local Search	60	1135	89	4	10259
LA26	20x10	1218	HybFlexGA		15	1249	31	6	12196
			LEKIN	EDD	1	1414	196	10	12853
				SPT	1	1900	682	8	16793
				General SB Routine	48	1387	169	15	13542
				Local Search	9	1218	0	0	11853
LA31	30x10	1784	HybFlexGA		14	1843	59	3	17502
			LEKIN	EDD	1	1913	129	3	17715
				SPT	1	2746	962	6	24571
				General SB Routine	280	1953	169	11	19044
				Local Search	4	1784	0	0	17340
LA36	15x15	1268	HybFlexGA		18	1360	92	9	19657
			LEKIN	EDD	1	1552	284	7	20848
				SPT	1	2298	1030	6	27542
				General SB Routine	28	1450	182	10	20385
				Local Search	122	1291	23	8	18724

- relativamente à coluna (6), a heurística *Local Search* obteve em sete instâncias $\overline{C}_{\max}$ igual ao valor óptimo (FT06, FT20, LA01, LA06, LA11, LA26 e LA31), o HybFlexGA obteve em três instâncias $\overline{C}_{\max}$ igual ao valor óptimo (FT06, LA01 e LA06), a heurística *General SB Routine* obteve numa instância $\overline{C}_{\max}$ igual ao valor óptimo (LA06). Pelo contrário, as regras de sequenciamento EDD e SPT não obtiveram em nenhuma das instâncias um valor de $\overline{C}_{\max}$ igual ao valor óptimo;
- nas colunas (7) e (8) estão registadas as médias do máximo atraso $\overline{T}_{\max}$ e as médias do somatório dos *jobs* em atraso $\overline{\sum U_i}$, estando estas directamente relacionados com os valores obtidos na coluna (6). Por conseguinte, nestas duas colunas, a heurística *Local Search* obteve valores nulos em sete instâncias, o HybFlexGA obteve valores nulos em três instâncias e a heurística *General SB Routine* obteve valores nulos numa instância. Mais uma vez, as regras de sequenciamento EDD e SPT não obtiveram bons resultados;
- na coluna (9) encontram-se registadas as médias do somatório dos tempos máximos de conclusão $\overline{\sum C_i}$. Os valores registados nesta coluna permitem-nos tirar conclusões quanto aos tempos de finalização das operações dos *jobs* nas máquinas, e consequentemente são, também, um indicador da forma como as operações dos *jobs* estão distribuídas ao longo do tempo nas máquinas. Da análise dos resultados desta coluna, concluímos que *Local Search* e HybFlexGA são os que apresentam melhores resultados.

Face aos resultados computacionais obtidos pelo HybFlexGA e LEKIN nas colunas (6), (7) e (8) da tabela 7.18, obtemos a grelha de classificação apresentada na tabela 7.19. Apesar do 2º lugar alcançado pelo HybFlexGA, os resultados apresentados na tabela 7.18 mostram que este obteve um desempenho melhor que a heurística *Local Search* nas instâncias FT06, FT10 e LA01. Porém, a heurística *Local Search* apresenta globalmente uma melhor performance.

Tabela 7.19 Classificação dos algoritmos de sequenciamento

Posição	Algoritmos de sequenciamento
1º	*Local Search*
2º	HybFlexGA
3º	*General SB Routine*
4º	EDD
5º	SPT

7.8 Resumo do Capítulo

Neste capítulo propusemos um modelo para a resolução de problemas de sequenciamento do tipo *Job-Shop*. O modelo proposto tem uma arquitectura composta por três módulos: interface com o utilizador, pré-processamento e sequenciamento. O algoritmo para a resolução de problemas de sequenciamento do tipo *Job-Shop* foi implementado no módulo de sequenciamento, sendo o referido algoritmo constituído por dois submódulos: decomposição e validação. Sucintamente, o submódulo de decomposição separa o problema *Job-Shop* original em vários problemas de uma só máquina e encontra as respectivas soluções óptimas ou quase-óptimas, sendo de seguida essas soluções integradas no problema principal. Muitas das vezes, o processo de integração das várias soluções no problema principal origina soluções inexequíveis. Assim, e sempre que isso acontece, o submódulo de validação ajusta as diferentes soluções obtidas para cada uma das máquinas no problema principal. O referido modelo proposto para a resolução de problemas de sequenciamento do tipo *Job-Shop* foi implementado e acrescentado ao software HybFlexGA.

Com o objectivo de ilustrar e clarificar o funcionamento do algoritmo proposto, este foi aplicado a três exemplos de problemas de sequenciamento do tipo *Job-Shop*: 5/3/*G*/C_{max} adaptado de Dileep [Dileep, 1996], 3/4/*G*/C_{max} adaptado de Pinedo [Pinedo, 1995] e 6/6/*G*/C_{max} proposto por Fisher e Thompson [Fisher e Thompson, 1963].

Relativamente aos resultados computacionais obtidos nas tabelas 7.15, 7.16 e 7.17, o HybFlexGA apresentou um bom desempenho em relação aos restantes algoritmos analisados, tendo encontrado a solução óptima num número significativo de instâncias. O HybFlexGA ficou em 2º lugar nos resultados computacionais obtidos na tabela 7.18, no entanto, e comparativamente à heurística *Local Search*, o HybFlexGA obteve um melhor desempenho nas instâncias FT06, FT10 e LA01. Contudo, a heurística *Local Search* apresenta globalmente uma melhor performance.

Podemos concluir que o HybFlexGA pode ser muito útil na resolução de problemas de sequenciamento do tipo *Job-Shop*. Assim, um dos trabalhos futuros poderá ser o desenvolvimento de uma versão industrial do HybFlexGA com o objectivo desta vir a ser utilizada em ambientes indústrias.

Para trabalho futuro deixamos, também, o problema de sequenciamento *Job-Shop* flexível (*flexible Job-Shop*). Neste tipo de problema existe mais do que uma máquina do mesmo tipo,

sendo permitido o processamento de uma operação em qualquer uma dessas máquinas, com o objectivo de minimizar o *makespan*. Este problema é também NP difícil porque para além do sequenciamento das operações existe também a necessidade de resolver o problema de afectação das operações às máquinas [Garey *et al.*, 1976], [Jansen *et al.*, 2000].

Capítulo 8 Conclusões

"*Não tenho nenhuma solução, mas é certo que admiro o problema*"

Ashleigh Brilliant

8.1 Introdução

Este capítulo apresenta, na secção 8.2, uma síntese conclusiva do trabalho desenvolvido ao longo da tese. Por fim, na secção 8.3 são referidas as perspectivas de trabalho futuro.

8.2 Síntese conclusiva

Cada capítulo apresenta no fim uma secção com um resumo do capítulo, onde é feita uma síntese de cada capítulo e onde são apresentadas algumas conclusões. Desta forma, apresenta-se a seguir uma síntese das principais conclusões do trabalho de doutoramento desenvolvido:

- Foram apresentados e discutidos muitos dos problemas associados à integração e ao controlo de robôs manipuladores e máquinas CNC industriais. Foram, também focadas, nesta tese, questões de acesso e comando remoto desses equipamentos. A filosofia adoptada pode ser usada noutros equipamentos de uma CFF, com o objectivo de desenvolver uma arquitectura que permita uma programação integrada de toda a célula. As potencialidades de funcionamento em rede (protocolo TCP/IP) das ferramentas de software e hardware desenvolvidas permitem a integração, a coordenação e o controlo de: robôs industriais, de máquinas CNC, de transportadores, entre outros, em SFF e CFF.
- As ferramentas de software desenvolvidas e apresentadas neste capítulo foram utilizadas com sucesso nos robôs e máquinas CNC que temos no laboratório (robôs: ABB IRB140, ABB IRB1400 e Scorbot ER VII; máquinas CNC: EMCO MILL 155 e EMCO TURN 55). De referir que, uma das ferramentas de software desenvolvida para os robôs

manipuladores industriais encontra-se, actualmente, em utilização intensiva numa empresa multinacional do ramo automóvel.

- Foi desenvolvida uma CFF com a finalidade de testarmos em aplicações industriais as potencialidades das ferramentas de software e hardware concebidas. Os resultados experimentais obtidos na CFF demonstram a viabilidade e o sucesso das ferramentas desenvolvidas na coordenação dos diferentes sectores utilizados na CFF, bem como no melhoramento das suas performances.
- Foram desenvolvidos operadores genéticos para melhorarem o desempenho dos algoritmos genéticos na resolução de problemas de sequenciamento de *jobs*. Os testes computacionais realizados demonstraram que o operador de cruzamento *two-point crossover: 4 children* e o operador de mutação *arbitrary 20%-job change* contribuem, de forma significativa, para a melhoria do desempenho dos algoritmos genéticos.
- Foi proposto um AG para a resolução de problemas de sequenciamento de *jobs* com uma só máquina. Os testes computacionais realizados às instâncias SMTWT (com 40, 50 e 100 *jobs*) demonstraram a eficiência do algoritmo proposto.
- Foi desenvolvido um modelo para a resolução de problemas de sequenciamento do tipo *Job-Shop*. O modelo proposto consiste em decompor o problema *Job-Shop* em vários problemas de uma só máquina. As soluções obtidas nos problemas de uma só máquina serão depois incorporadas no problema principal. Os resultados computacionais obtidos, comparativamente aos restantes algoritmos de sequenciamento analisados, mostraram a eficiência do modelo proposto.
- Foi concebida a ferramenta de software HybFlexGA com a finalidade de reunir todos os modelos de sequenciamento desenvolvidos ao longo do trabalho de doutoramento. O HybFlexGA permite, também, a resolução de problemas de sequenciamento no âmbito da CFF desenvolvida.

8.3 Perspectivas de trabalho futuro

Relativamente ao trabalho realizado há aspectos que podem ser melhorados ou explorados, o que permite perspectivar a possibilidade de futuros desenvolvimentos nas seguintes vertentes:

- Desenvolvimento de aplicações industriais que utilizem robôs manipuladores de diferentes configurações (*e.g.*, robôs esféricos, cartesianos, cilíndricos, entre outras) e máquinas CNC de diferentes tipos (*e.g.*, máquinas de corte por laser, máquinas de corte

de tecido automáticas, máquinas de medição de coordenadas (CMM), centros de furação, entre outras).

- Aplicar e testar a arquitectura do software desenvolvida a outros equipamentos que possam fazer parte de uma CFF ou SFF, como por exemplo, sistemas de visão por computador, autómatos programáveis, AGV, entre outros.
- Desenvolvimento de uma versão industrial do HybFlexGA com o objectivo desta poder ser utilizada em ambientes industriais.

Referências

[ABB, 2003a] ABB Manual do Produto IRB2400, ABB Robotics Products AB, 2003.

[ABB, 2003b] ABB IRB2400 Users Manual, ABB Robotics AB, 2003.

[ABB, 2003c] ABB Manual de Referência RAPID, ABB Robotics AB, 2003.

[ABB, 2003d] ABB IRB140 Users Manual, ABB Robotics AB, 2003.

[ABB, 2003e] ABB IRB1400 Users Manual, ABB Robotics AB, 2003.

[Abdul Razaq *et al.*, 1990] Abdul Razaq T. S., Potts C. N. and Van Wassenhove L. N., "A survey for the single-machine scheduling total weighted tardiness scheduling problem", Discrete Applied Mathematics, nº 26, pp. 235-253, 1990.

[Adams *et al.*, 1988] J. Adams, E. Balas, D. Zawack, "The shifting bottleneck procedures for job shop scheduling", Management Science, vol. 34, nº 3, USA, pp. 391-401, 1988.

[Alves, 2002] William P. Alves, "C++ Builder 6", Erica, 2002.

[Amico *et al.*, 1993] Dell's Amico, M. Turbian, "Applying tabu search to the job shop scheduling problem", Annual Operation Research, 40, pp. 231-252, 1993.

[Arbita *et al.*, 1997] Arbita, A. and S. E. Elmaghraby, "The planning and scheduling of production systems", Chapman & Hall, 1997.

[Arentoft *et al.*, 1991] Arentoft M., Fuchs J., Parrod Y., Stader J., Stokes I., e Vadon H., "Optimum-AIV: A planning and scheduling system for spacecraft AIV", NASA. Goddard Space Flight Center, The 1991 Goddard Conference on Space Applications of Artificial Intelligence, pp. 3-13, 1991.

[Back, 1996] Back T., "Evolutionary Algorithms in Theory and Practice", Oxford University Press, 1996.

[Baker, 1974] Baker, K. R., "Introduction to Sequencing and Scheduling", Wiley, New York, 1974.

[Banzhaf *et al.*, 1998] W. Banzhaf, P. Nordin, et al. "Genetic Programming: an introduction", Morgan Kaufmann, 1998.

[Blazewicz *et al.*, 1994] Blazewicz, J., Ecker, K. H., Pesch, E., Smith, G. and Weglarz, J., "Scheduling in Computer and Manufacturing Systems", Springer, 2nd edition, New York, 1994.

[Blazewicz *et al.*, 2001] Blazewicz, J., Ecker, K. H., Pesch, E., Smith, G. and Weglarz, J., "Scheduling Computer and Manufacturing Processes", Springer, 2nd edition, New York, 2001.

[Brucker, 2001] Brucker P., "Scheduling algorithms", Springer, 3rd edition, New York, 2001.

[Byrkett *et al.*, 1988] D. L. Byrkett, M. H. Ozeden and J. M. Patton, "Integrating flexible manufacturing systems with traditional manufacturing, planning, and control", Journal of Production and Inventory Management, 29, pp. 15-21, 1988.

[Calvert, 1997] Charlie Calvert, "Borland C++ Builder", SAMS Publishing, 1997.

[Camarinha-Matos e Afsarmanesh, 1999] Camarinha-Matos L. M. and Afsarmanesh H., "The Virtual Enterprise Concept", in IFICTC5/PRODNET Working Conf. Infraestructures for Virtual Enterprises, Kluwer Academic Publishers, 1999.

[Carlier e Pinson, 1989] Carlier J. and E. Pinson, "An algorithm for solving the job-shop problem", Manage, Sci., 35, 2, pp. 164-176, 1989.

[Chan *et al.*, 2002] F. T. S. Chan, H. K. Chan and H. C. W. Lau, "The stat of the art in simulation study on FMS scheduling: comprehensive survey", International Journal of Advanced Manufacturing Technology, pp. 830-849, 2002.

[Chang, 1991] T. C. Chang, R. A. Wysk, and H. P. Wang, "Computer-Aided Manufacturing", Englewood Cliffs, NJ: Prentice-Hall, 1991.

[Cho *et al.*, 1995] H. Cho, T. K. Kumaran, and R. A. Wysk, "Graph-Theoretic Deadlock and Resolution for Flexible Manufacturing Systems", IEEE Trans. on Robotics and Automation, vol. 11, nº. 3, pp. 413–421, June 1995.

[Chryssolouris, 1987] Chryssolouris G., "MADEMA: An Approach to Intelligent Manufacturing Systems", CIM Review, Volume 3, N.º 3, pp. 11-17, 1987.

[Colorni *et al.*, 1994] A. Colorni, M. Dorigo, V. Maniezzo, M. Trubian, "Ant system for Job-shop Scheduling", JORBEL-Belgian Journal of Operations Research, Statistics and Computer Science, 34(1), pp. 39-53, 1994.

[Congram *et al.*, 1988] Congram, Richard, Potts C. N. and Velde S. L. V., "An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem", Preprint Series nº OR95, University of Southampton, 1998.

[Conway *et al.*, 1967] Conway, R. W., Maxwell, W. L., and Miller, L. W., "Theory of Scheduling", Addison-Wesley, Publishing Company, 1967.

[Cormen, 1990] Cormen T. H., Leiserson C. E., Rivest R. L. "Introduction to Algorithms", MIT Press,1990.

[Costa e Simões, 2004] Ernesto Costa e Anabela Simões, "Inteligência Artificial, Fundamentos e Aplicações", FCA-Editora de Informática, 2004.

[Crauwels *et al.*, 1998] Crauwels H. A. J., Potts C. N. and Van Wassenhove L. N., "Local Search heuristics for the single machine total weighted tardiness scheduling problem", Informs Journal on Computing, vol. 10, nº 3, pp. 341-350, 1998.

[Dauzére-Pérés *et al.*, 1993] Dauzére-Pérés, S. Lassere J. B., "A Modified Shifting Bottleneck Procedure for Job Shop Scheduling", International Journal of Production Research, April 31, pp. 923-932, 1993.

[Davis, 1991] Lawrence Davis, "Handbook of genetic algorithms", Van Nostrand Reinhold, New York, 1991.

[Dileep, 1996] Dileep R. Sule, "Industrial Scheduling", PWS Publishing Company, pp. 152, 1996.

[Dimopoulos e Zalzala, 2000] Dimopoulos C. E Zalzala M. S. "Recent Developments in Evolutionary Computation for Manufacturing Optimization: Problems, Solutions and Comparisons", IEEE Transactions on Evolutionary Computation, n.º 4, Volume 2, pp. 93-113, 2000.

[Drabble *et al.*, 1992] Drabble, Brian, Kirby, Richard, Tate, Austin, "O-Plan2 – The Open Planning architecture, NASA. Ames Research Center", Working Notes from the 1992 AAAI Spring Symposium on Practical Approaches to Scheduling and Planning, pp. 87-91, 1992.

[Eiben e Smith, 2003] A.E. Eiben e J.E. Smith, "Introduction to Evolutionary Computing", Spring Verlag, 2003.

[Emco, 2001] Emco Mill 155, Machine Description, 2001.

[Emco, 2002] Emco Turn 55, Machine Description, 2002.

[Eshed Robotec, 1995a] Advanced Terminal Software, Reference Guide", Eshed Robotec, 1ª Edição, 1995.

[Eshed Robotec, 1995b] ACLoff-line, User's Manual, Eshed Robotec, 2ª Edição, 1995.

[Eshed Robotec, 1995c] Advanced Control Language, Reference Guide, Eshed Robotec, 4ª Edição, 1995.

[Eshed Robotec, 1996] Scorbot ER VII, User's Manual, Eshed Robotec, 2ª Edição, 1996.

[Fa e Lin, 2003] Cheng-Fa Tsai and Feng-Cheng Lin, "A New Hybrid Heuristic Technique for Solving Job-shop Scheduling Problem", Proceedings of the IEEE Int. Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, pp. 53-58, 2003.

[Ferrolho e Crisóstomo, 2003] António Ferrolho e Manuel Crisóstomo, "Software Development to Control the Scorbot ER VII Robot With a PC", Proceedings of the 5th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering (MMACTEE 2003)", Vouliagmeni, Athens, Greece, December 2003, em CD-ROM.

[Ferrolho e Crisóstomo, 2004a] António Ferrolho e Manuel Crisóstomo, "Software Development to Control the Scorbot ER VII Robot With a PC", WSEAS Transactions on Circuits and Systems. Issue 2, Volume 3, April 2004, ISSN: 1109-2734, pp. 247-253.

[Ferrolho e Crisóstomo, 2004b] António Ferrolho e Manuel Crisóstomo, "Development of a Flexible Manufacturing Cell", Proceedings of the 3th WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA2004), Salzburg, Austria, February 2004, em CD-ROM.

[Ferrolho e Crisóstomo, 2004c] António Ferrolho e Manuel Crisóstomo, "Development of a Flexible Manufacturing Cell", WSEAS Transactions on Electronics. Issue 2, Volume 1, April 2004, ISSN: 1109-9445, pp. 404-409.

[Ferrolho e Crisóstomo, 2004d] António Ferrolho e Manuel Crisóstomo, "Genetic Algorithms: concepts, techniques and applications", Proceedings of the WSEAS International Conference on Engineering Education (EE2004), Venice, Italy, November 2004, em CD-ROM.

[Ferrolho e Crisóstomo, 2005a] António Ferrolho and Manuel Crisóstomo, "Flexible Manufacturing Cell: Development, Coordination, Integration and Control", Proceedings of the IEEE 5th International Conference on Control and Automation, Budapest, Hungary, June 2005, pp. 1050-1055.

[Ferrolho e Crisóstomo, 2005b] António Ferrolho and Manuel Crisóstomo, "Scheduling and Control of Flexible Manufacturing Cells Using Genetic Algorithms", WSEAS Transactions on Computers. Issue 6, Volume 4, June 2005, ISSN: 1109-2750, pp. 502-510.

[Ferrolho e Crisóstomo, 2005c] António Ferrolho and Manuel Crisóstomo, "Genetic Algorithms for Solving Scheduling Problems in Flexible Manufacturing Cells", Proceedings of the 4th WSEAS International Conference on Electronics, Signal Processing and Control (ESPOCO2005), Rio de Janeiro, Brazil, April 2005, em CD-ROM.

[Ferrolho e Crisóstomo, 2005d] António Ferrolho e Manuel Crisóstomo, "Genetic Algorithms: concepts, techniques and applications", WSEAS Transactions on Advances in Engineering Education. Issue 1, Volume 2, January 2005, ISSN: 1790-1979, pp. 12-19.

[Ferrolho e Crisóstomo, 2006a] António Ferrolho e Manuel Crisóstomo, "A New Concept of Genetic Operators for Scheduling Problems", Proceedings of the IEEE 4th International Conference on Computational Cybernetics, Tallinn, Estonia, August 20-22, 2006, pp. 131-136.

[Ferrolho e Crisóstomo, 2006b] António Ferrolho e Manuel Crisóstomo, "Control and Scheduling in Flexible Manufacturing Cells", Proceedings of the IEEE International Conference on Industrial Technology (ICIT06), Mumbai, India, December 15-17, 2006, ISBN: 1-4244-0726-5, pp. 1241-1246.

[Ferrolho e Crisóstomo, 2006c] António Ferrolho e Manuel Crisóstomo, "Scheduling Jobs in Flexible Manufacturing Cells with Genetic Algorithms", Research in Computing Science, Special issue: Advances in Computer Science and Engineering, Vol. 23, ISSN: 1870-4069, 2006, pp. 31-40.

[Ferrolho e Crisóstomo, 2006d] António Ferrolho e Manuel Crisóstomo, "Scheduling Jobs in Flexible Manufacturing Cells with Genetic Algorithms", poster session in 15th International Conference on Computing (CIC2006), Mexico City, Mexico, November 21-24, 2006.

[Ferrolho e Crisóstomo, 2006e] António Ferrolho e Manuel Crisóstomo, "Genetic Algorithm for the Single Machine Total Weighted Tardiness Problem", Proceedings of the IEEE International Conference on E-Learning in Industrial Electronics (ICELIE06), Hammamet, Tunisia, December 18-20, 2006, ISBN: 1-4244-0324-3, pp. 17-22.

[Ferrolho e Crisóstomo, 2007a] António Ferrolho e Manuel Crisóstomo, "Intelligent Control and Integration Software for Flexible Manufacturing Cells", IEEE Transactions on Industrial Informatics, Vol. 3, no. 1, ISSN: 1551-3203, pp. 3-11, February 2007.

[Ferrolho e Crisóstomo, 2007b] António Ferrolho e Manuel Crisóstomo, "Control and Scheduling Software for Flexible Manufacturing Cells", Industrial Robotics: Programming, Simulation and Applications, ISBN: 3-86611-286-6, pp. 315-340, Advanced Robotic Systems, 2007.

[Ferrolho e Crisóstomo, 2007c] António Ferrolho e Manuel Crisóstomo, "Single Machine Total Weighted Tardiness Problem with Genetic Algorithms", Proceedings of the IEEE International Conference on Computer Systems and Applications (AICCSA07), Amman, Jordania, May 13-16, 2007, em CD-ROM.

[Ferrolho *et al.*, 2005] António Ferrolho, Manuel Crisóstomo and Miguel Lima, "Intelligent Control Software for Industrial CNC Machines", Proceedings of the IEEE 9th International Conference on Intelligent Engineering Systems, Cruising on Mediterranean Sea, September 2005, em CD-ROM.

[Ferrolho *et al.*, 2006a] António Ferrolho, Manuel Crisóstomo e Miguel Lima, "Scheduling in Flexible Manufacturing Cells", Recent Advances in Control Systems, Robotics and Automation, International Society for Advanced Research, 2006, pp. 93-99. Editor: Salvatore Pennacchio, publicado por Internationalsar. ISBN: 88-901928-0-1.

[Ferrolho *et al.*, 2006b] António Ferrolho, Manuel Crisóstomo e Miguel Lima, "Scheduling in Flexible Manufacturing Cells", International Journal of Factory Automation, Robotics and Soft Computing, Issue 2, April 2006, ISSN: 1828-6984, pp. 56-62.

[Ferrolho, 2001] António Ferrolho, "Desenvolvimento de uma Células de Fabrico Flexível", Tese de Mestrado, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 2001.

[Figueiredo, 2002] Ana M. N. A. Baptista Figueiredo, "Análise e Desenvolvimento de Mecanismos e Algoritmos de Apoio ao Escalonamento Orientado ao Produto", Tese de Doutoramento, Universidade do Minho, 2002.

[Fisher e Thompson, 1963] Fisher H. and Thompson G. L., "Probabilistic learning combinations of local Job Shop scheduling rules", Industrial Scheduling, Prentice Hall, Englewood Cliffs, NJ, pp. 225-251, 1963.

[Fogel *et al.*, 1966] Fogel L. J., Owens A. J. e Walsh M. J., "Artificial intelligence through simulated evolution", John Wiley and Sons, New York, 1966.

[Fogel, 1995] Fogel D.B., "Evolutionary Computation", IEEE Press, 1995.

[Fogel, 1998] Fogel D.B. "An Evolutionary Approach to the Travelling Salesman Problem", Biol. Cybernet, 1998.

[Foo e Takefuji, 1988] Yoon-Pin Simon Foo and Yoshiyasu Takefuji, "Stochastic Neural Networks for Solving job-shop scheduling part 1: Problem representation", IEEE International Conference on Neural Networks, vol. 2, pp. 275-282, 1988.

[Fox e Sycara, 1990] Fox, M. and Sycara, K., "The CORTES Project: A Unified Framework for Planning, Scheduling and Control", Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA, 1990.

[French, 1982] French, S., "Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop", Ellis Horwood, Chichester, 1982.

[Gaither, 1989] Gaither, N., "Production and Operations Management", Dryden Press, 1989.

[Garey *et al.*, 1976] Garey, M. R., Johnson, D. S. e Sethi, R., "The complexity of Flow-Shop and Job-Shop Scheduling", Mathematics of Operations Research, pp. 117-129, 1976.

[Garey *et al.*, 1979] Garey, Michael R. and Johnson, David S., "Computers and Intractability – A guide to the theory of NP-Completeness", W. H. Freeman and Company, San Francisco, 1979.

[Goldberg, 1989] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

[Goldberg, 2002] D. E. Goldberg, "The Design of Innovation: lessons from and for competent genetic algorithms", Kluwer Academi Press, 2002.

[Goldratt e Fox, 1986] Goldratt E. and Fox R., "The Race", North River Press, 1986.

[Grady e Menon, 1986] P. J. O'Grady and U. Menon, "A concise review of flexible manufacturing systems and FMS literature", Computers in Industry, 7, pp. 155-117, 1986.

[Graves, 1981] Graves, S. C., "A review of production scheduling", Operations research, 29 (4), pp.646-675, 1981.

[Hadj-Alouane *et al.*, 1988] N. B. Hadj-Alouane, J. Chaar, A. Naylor, and R. A. Volz, "Material handling systems as software components: An implementation," Tech. Paper, Center for Res. on Integrated Manuf., The Univ. Michigan, May 1988.

[Holland, 1975] Holland, J., "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975.

[Hong e Jian, 2006] Zhou Hong and Wang Jian, "A Cooperative Coevolutionary Algorithm with Application to Job Shop Scheduling Problem", Proceedings of the IEEE 2006 International Conference on Service Operations and Logistics, and Informatics, Shanghai, pp. 746-751, 2006.

[Jackson, 1956] Jackson, J. R., "An extension of Johnson's results on job Lot Scheduling", Naval Research Logistics Quarterly, pp. 201-203, 1956.

[Jain *et al.*, 1999] Jain, A. S. e Meeran, S., "Deterministic job-shop scheduling: Past, present and future", European Journal of Operational Research, pp. 390-434, 1999.

[Jansen *et al.*, 2000] Jansen, K., Mastrolilli, M. e Solis-Oba, R., "Approximation Algorithms for Flexible Job Shop Problems", Proceedings of Latin American Theoretical Informatics (LATIN, 2000), pp. 68-77, 2000.

[Joel, 1972] Joel Shwimer, "On the n-job, one-machine, sequence-independent scheduling problem with tardiness penalties: a branch-bound solution", Management Science, vol. 18, nº 6, pp. 301–313, 1972.

[Johnson e Miller, 1994] Johnson, Mark and Miller, Glenn, "SPIKE: Intelligent Scheduling of

Hubble Space Telescope Observations", Intelligent Scheduling, ed. M. Fox and M. Zweben, Morgan-Kaufmann, ISBN 1-55860-260-7, pp. 391-422, 1994.

[Joshi e Mettala, 1991] S. Joshi, R. A. Wysk, and E. G. Mettala, "Automatic Generation of Control System Software for Flexible Manufacturing Systems", IEEE Trans. on Robotics and Automation, vol. 7, no. 6, pp. 283–291, December 1991.

[Joshi *et al.*, 1995a] S. B. Joshi, Erik G. Mettala, Jeffrey S. Smith, and Richard A. Wysk, "Formal Models for Control of Flexible Manufacturing Cells: Physical and System Model", IEEE Trans. on Robotics and Automation, vol. 11, no. 4, pp. 558–570, August 1995.

[Joshi *et al.*, 1995b] S. B. Joshi, J. S. Smith, R. A. Wysk, B. Peters, and C. Pegden "Rapid-CIM: An approach to rapid development of control software for FMS control", 27th CIRP International Seminar on Manufacturing Systems, Ann Arbor, MI, 1995.

[Kaltwasser *et al.*, 1986] J. Kaltwasser, A. Hercht and R. Lang, "Hierarchical control of flexible manufacturing systems", IFAC Information Control Problems in Manufacturing Technology, Suzdal, USSR, pp. 37-44, 1986.

[Keefe e Kasirajan, 1992] R. M. O'Keefe and T. Kasirajan, "Interaction between dispatching and next station selection rules in a dedicated flexible manufacturing system", International Journal of Production Research, 30(8), pp. 1753-1772, 1992.

[Kolonko, 1999] M. Kolonko, "Some new results on simulated annealing applied to the job shop scheduling problem", European Journal of Operational Research, pp. 123-136, 1999.

[Kruglinski, 1997] David J. Kruglinski, "Visual C++", Microsoft Press, 1997.

[Kusiak, 1986] A. Kusiak, "Modelling and Design of Flexible Manufacturing Systems", Elsevier Science Publishers, 1986.

[Kutanoglu e Sabuncuoglu, 1999] Kutanoglu E. e Sabuncuoglu I., "Na Analysis of heuristics in a dynamic job shop with weighted tardiness objectives", International Journal of Production Research, n.º 1, Volume 37, pp. 165-187, 1999.

[Lawler *et al.*, 1993] Lawler, E. L., Lenstra, J. K., e Rinnooy Kan, A. H. G., "Sequencing and Scheduling: Algorithms and Complexity" in Handbook in Operations Research and Management Science 4: Logistics of Production and Inventory, 1993.

[Lawler, 1977] Lawler E. L., "A pseudopolinomial algorithm for sequencing jobs to minimize total tardiness", Annals of Discrete Mathematics, pp. 331-342, 1977.

[Lawler, 1983] Lawler, E. L., "Mathematical Programming: The State of the Art", Springer Verlag, 1983.

[Lawrence, 1984] Lawrence S., "Resource constrained project scheduling: an experimented investigation of heuristic scheduling techniques", graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, PA, 1984.

[Li, 1994] L. Li, "Object-Oriented Modeling and Implementation of Control Software for a Robotic Flexible Manufacturing Cell", Robotics and Computer Integrated

Manufacturing, 1994.

[Lui e Xi, 2006] Lin Lui and Yugeng Xi, "A Hybrid Genetic Algorithm for Job Shop Scheduling Problem to Minimize Makespan", Proceedings of the IEEE 6th World Congress on Intelligent Control and Automation, June 21-23, Dalian, China, pp. 3709-3713, 2006.

[Madureira *et al.*, 2001] Madureira A. M., Ramos C., Silva S. C., "A GA based scheduling system for dynamic single machine problem", Proceedings of the 4th IEEE International Symposium on Assembly and Task Planning Soft Research Park, Fukuoka, Japan, 2001.

[Madureira, 1999] Madureira A. M., "Meta-heuristics for the single machine scheduling total weighted tardiness problem", International Symposium on Assembly and Task Planning (ISATP 1999), Porto, Portugal, 1999.

[Madureira, 2003] Ana M. D. Madureira, "Aplicação de Meta-Heurísticas ao Problema de Escalonamento em Ambiente Dinâmico de Produção Discreta", Tese de Doutoramento, Universidade do Minho, 2003.

[Magy e Haidegger, 1993] E. Magy and G. Haidegger, "Object-Oriented Approach for Analyzing, Designing and Controlling Manufacturing Cells" Proceedings of AUTOFACT'93, Chicago, 1993.

[Maimon e Fisher, 1988] 0. Z. Maimon and E. L. Fisher, "An object-based representation method for a manufacturing cell controller," Artijcial Intelligence in Engineering, vol. 3, no. 1, 1988.

[Manderick e Spiessens, 1994] B. Manderick and P. Spiessens, "How to select genetic operators for combinatorial optimization problems by analyzing their fitness landscape", Computational Intelligence Imitating Life, IEEE Press, New York, pp.170-181, 1994.

[Mastrolilli, 1998] Mastrolilli M., Gambardella L. M., "Effective Neighborhood Functions for the Flexible Job Shop Problem", Journal of Scheduling, Volume 3, Issue 1, pp. 3-20, 1998.

[Michalewicz, 1996] Michalewicz Zbigniew, "Genetic Algorithms + Data Structures = evolution programs", Third Revised and Extended Edition, Springer – Verlag Berlin Heidelberg, New York, 1996.

[Mikkel, 2003] Mikkel T. Jensen, "Generating Robust and Flexible Job Shop Scheduling Using Genetic Algorithms", IEEE Transactions on Evolutionary Computation, vol. 7, nº. 3, June 2003, pp. 275-288, 2003.

[Mitchell, 1996] M. Mitchell, "An introduction to genetic algorithms", MIT Press, 1996.

[Morton *et al.*, 1993] Morton, E. Thomas and Pentico, David W., "Heuristic Scheduling Systems", John Wiley & Sons, 1993.

[Murata *et al.*, 1994] T. Murata and H. Ishibuchi, "Performance Evaluation of Genetic Algorithms for Flowshop Scheduling Problems", Proceedings of the 1st IEEE Conference on Evolutionary Computation, Orlando, USA, June 27-29, pp.812-

817, 1994.

[Murata *et al.*, 1996] T. Murata and H. Ishibuchi, "Positive and Negative Combination Effects of Crossover and Mutation Operators in Sequencing Problems", Proceedings IEEE Int. Conf. Evol. Computation, Piscataway, pp. 170-175, 1996.

[Nilsson, 1992] K. Nilsson, "Application Oriented Programming and Control of Industrial Robots", M.Sc. Thesis, July 1992.

[Nilsson, 1996] K. Nilsson, "Industrial Robot Programming", Ph.D. Thesis, Department of Automatic Control, Lund Institute of Technology, May of 1996.

[Nowicki e Smutnicki, 1996] E. Nowicki and C. Smutnicki, "A Fast Taboo Search Algorithm for the Job Shop Problem", Management Science, vol. 42, pp. 797-813, 1996.

[Oliver *et al.*, 1987] J. Oliver, D. Smith and J. Holland, "A study of permutation crossover operators on the traveling salesman problem", Proceedings of the Second ICGA, pp. 224-230, 1987.

[Peter *et al.*, 1997] Peter A. Huegler and Francis J. Vasko, "A performance comparison of heuristics for the total weighted tardiness problem", Computers & Industrial Engineering, vol. 32, nº 4, pp. 753–767, 1997.

[Pinedo e Chao, 1998] Pinedo, M. e Chao, X., "Operations Scheduling with Applications in Manufacturing and Services", McGraw Hill, 1998.

[Pinedo, 1995] Pinedo M., "Scheduling: theory, algorithms and systems", 1ª edição, Prentice-Hall, 1995.

[Pinedo, 2002] Pinedo M., "Scheduling: theory, algorithms and systems", 2ª edição, Prentice-Hall, 2002.

[Pires, 2005] Eduardo J. Solteiro Pires, "Uma Perspectiva Evolutiva dos Sistemas Robóticos", Tese de Doutoramento, Universidade de Trás-os-Montes e Alto Douro, 2005.

[Potts e Wassenhove, 1991] Potts C. N. and Van Wassenhove L. N., "Single machine tardiness sequencing heuristics", IIE Transactions, vol. 23, nº 4, pp. 346-354, 1991.

[Potts *et al.*, 1985] C. N. Potts and L. N. Van Wassenhove, "A branch and bound algorithm for the total weighted tardiness problems", Operations research, vol. 33, nº 2, pp. 363–377, 1985.

[Rabelo e Camarinha-Matos, 1994] Rabelo, R. e Camarinha-Matos, L.M., "Negotiation in Multiagent Based Dynamic Scheduling", International Journal on Robotics and Computer Integrated Manufacturing, vol. 11, nº 4, pp.303-310, Pregamon, 1994.

[Rabelo, 1997] Rabelo, R.J., "Um enquadramento para o Desenvolvimento de Sistemas de Escalonamento Ágil da Produção – Uma abordagem MultiAgente", Tese de Doutoramento, Universidade Nova de Lisboa, 1997.

[Ramos, 2001] Carlos Ramos, "Introdução aos Algoritmos Genéticos", Revista da Universidade Moderna do Porto, colecção Ciências de Engenharia, 2001.

[Ranky, 1990] Ranky, P. "FMS cell and system operation control strategies and solutions –

Flexible Manufacturing Cells and Systems in CIM", CIMware Limited, England, 1990.

[Rechenberg, 1973] Rechenberg I., "Evolutionstratgie: optimierung technischer susteme nach prinzipien der biologischen evolution", Formmann-Holzboog Verlag, Stuttgart, 1973.

[Reisdorph e Henderson, 1997] Reisdorph K. and Henderson K., "Borland C++ Builder", SAMS Publishing, 1997.

[Rinnooy *et al.*, 1975] H. G. Rinnooy Kan, B. J. Lageweg, J. K. Lenstra, "Minimizing total costs in one-machine scheduling", Operations research, vol. 23, nº 5 pp. 908–927, 1975.

[Sarin e Chen, 1987] S. C. Sarin and C. S. Chen, "The machine loading and toll allocation problem in a flexible manufacturing system", International Journal of Production Research, 25, pp. 1081-1094, 1987.

[Sauer, 1991] Sauer Jürgen, "Knowledge Based Scheduling in PROTOS", Proceedings of IMACS World Congress on Computation and Applied Mathematics, Vichnevetski, R. (ed.), College, Dublin, 1991.

[Schrage *et al.*, 1978] L. Schrage and K. R. Baker, "Dynamic programming solution of sequencing problem with precedence constraints", Operations research, vol. 26 pp.444–449, 1978.

[Schwefel, 1981] Schwefel Hans-Paul, "Numerical Optimization of Computer Models", John Wiley and Sons, New York, 1981.

[Smith, 1994] Smith S. F., "Intelligent Scheduling", Morgan Kaufman Publishers, San Francisco, 1994.

[Sule, 1997] Sule, D. R., "Industrial Scheduling", PWS Publishing, 1997.

[Syswerda, 1991] G. Syswerda, "Scheduling optimization using genetic algorithms", in L. Davis (Ed.) Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, pp.332-349, 1991.

[Van *et al.*, 1992] Van Laarhoven PJM, Aarts EHL and Lenstra JK, "Job Shop scheduling by simulated annealing", Operations Research, 40, pp. 113-125, 1992.

[Viswanadham e Narahari, 1992] N. Viswanadham and Y. Narahari, "Performance Modeling of Automated Manufacturing Systems", Prentice-Hall, INC., 1992.

[Waldner, 1992] J. B. Waldner, "CIM, Principles of Computer Integrated Manufacturing", John Wiley & Sons, 1992.

[Wang e Zheng, 2001] L. Wang and D. Z. Zheng, "Na effective hybrid optimization strategy for Job-shop scheduling problems", Computers & Operations Research 28, pp. 585-596, 2001.

[Wang e Zheng, 2002] L. Wang and D. Z. Zheng, "A Modified Genetic Algorithm for Job Shop Scheduling", The International Journal of Advanced Manufacturing Technology by Springer-Verlag, vol. 20, pp. 72-76, 2002.

[Wu *et al.*, 2004] C. G. Wu, X. L. Xing, H. P. Lee, C. G. Zhou, and Y. C. Liang, "Genetic Algorithm Application on the Job Shop Scheduling Problem", in Proc. of the IEEE Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004, pp. 2102-2106, 2004.

[Xia *et al.*, 2004] Xia Weijun, Wu Zhiming, Zhang Wei and Yang Genke "A New Hybrid Optimization Algorithm for the Job-shop Scheduling Problem", Proceedings of the IEEE 2004 American Control Conference, Boston, Massachusetts, June 30 – July 2, pp. 5552-5557, 2004.

[Yang e Wang, 2000] S.Yang and D.Wang, "Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling," in IEEE Trans. On Neural Networks, Vol. 11, No. 2, pp. 474-486, 2000.

[Yang, 2005] Shengxiang Yang, "An improved adaptive neural network for job-shop scheduling", Proceedings of the IEEE 2005 International Joint Conference on Systems, Man and Cybernetics, Vol. 2, pp. 1200-1205, 2005.

[Yang, 2006] Shengxiang Yang, "Job-Shop Scheduling with an Adaptive Neural Network and Local Search Hybrid Approach", Proceedings of the IEEE 2006 International Joint Conference on Neural Networks, July 16-21, Vancouver, Canada, pp. 2720-2727, 2006.

Sítios WWW

http#1: http://tamcam.tamu.edu/rapidcim/rc.htm

http#2: http://www.engr.psu.edu/cim/

http#3: http://www.engr.psu.edu/cim/FAME/CIMLAB/cim_home.html

http#4: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html

http#5: http://www.stern.nyu.edu/om/software/lekin

http#6: http://ise.ksc.nasa.gov/processmodelforum/presentations0005/tsld003.htm

http#7: http://www-2.cs.cmu.edu/~sycara/cora.html

Apêndice A

População inicial e instâncias utilizadas

A.1 População inicial utilizada nos testes computacionais

A tabela A.1 apresenta a população inicial (Ψ_t) utilizada nos testes computacionais, realizados na avaliação dos operadores genéticos. Esta população, gerada aleatoriamente no HybFlexGA, é constituída por 100 cromossomas (N_{pop}=100), possuindo cada cromossoma 40 *jobs* (n=40). Cada cromossoma representa uma possível solução para o problema, não tendo *jobs* repetidos ou ausentes.

Tabela A.1 População inicial utilizada nos testes computacionais

x_0^1	6	32	29	13	39	10	24	3	30	37	31	2	7	38	23	21	8	11	19	9	40	34	15	1	4	26	14	12	5	36	28	20	18	22	25	35	27	33	17	16
x_0^2	19	10	38	8	33	3	24	11	27	9	15	6	30	1	36	12	16	23	28	4	5	21	25	31	2	20	40	14	26	18	34	32	29	7	17	35	22	39	13	37
x_0^3	21	4	18	29	10	35	9	13	33	31	14	38	8	3	16	19	34	25	36	27	2	26	15	32	22	24	23	12	5	6	17	28	40	1	37	20	11	39	30	7
x_0^4	31	2	8	6	9	5	29	10	23	7	16	39	37	17	12	30	28	14	1	13	25	15	27	21	34	20	35	32	26	22	38	24	4	40	36	3	33	18	11	19
x_0^5	7	30	20	12	5	4	1	8	17	24	35	32	18	22	23	15	13	39	40	14	37	11	33	6	9	16	2	26	29	36	34	19	10	38	31	3	27	21	25	28
x_0^6	26	40	2	4	25	6	13	37	9	10	31	5	1	39	17	30	19	14	35	11	12	8	3	24	29	32	21	28	33	36	23	22	20	7	15	27	34	38	16	18
x_0^7	4	6	12	9	38	19	7	8	2	23	24	30	40	13	1	39	36	18	17	29	26	15	5	34	21	31	32	16	33	22	37	11	27	3	10	20	14	28	35	25
x_0^8	18	14	23	2	13	3	39	9	21	31	8	17	6	15	10	34	7	29	36	40	35	33	12	20	11	30	25	32	22	5	16	24	26	19	27	4	37	28	38	1
x_0^9	1	17	14	33	11	6	37	22	10	21	34	19	23	24	32	20	28	4	38	7	2	39	25	35	5	3	16	40	27	9	26	36	15	29	30	8	31	18	13	12
x_0^{10}	38	23	1	28	15	35	16	10	4	40	9	18	36	24	27	25	30	20	21	3	29	6	7	34	2	26	14	32	17	13	39	31	11	22	8	33	37	12	5	19
x_0^{11}	26	2	28	11	35	8	34	12	39	38	9	10	21	19	22	31	18	17	30	16	5	4	24	37	40	6	23	29	14	15	36	33	7	25	3	20	32	1	27	13
x_0^{12}	1	3	5	12	26	38	8	13	11	32	35	6	4	24	20	34	16	2	10	19	18	7	39	33	15	23	21	30	17	25	22	9	14	36	40	37	27	29	28	31
x_0^{13}	1	4	6	17	25	9	10	32	13	8	5	20	11	15	26	12	40	18	36	22	7	35	30	2	29	27	24	28	19	23	31	3	33	34	39	21	38	37	14	16
x_0^{14}	40	6	15	25	28	9	1	37	16	23	20	21	14	34	29	17	3	26	36	35	31	18	10	27	2	7	33	5	11	38	12	8	13	22	30	39	24	19	4	32
x_0^{15}	23	38	9	34	13	4	37	11	14	12	29	1	7	8	31	28	20	3	39	18	5	33	26	27	15	19	10	30	40	21	25	17	35	24	6	22	2	16	32	36
x_0^{16}	15	10	11	37	33	3	21	7	20	17	32	4	29	24	36	16	31	1	19	28	26	35	6	2	14	9	40	12	30	23	5	13	25	34	39	8	38	18	27	22
x_0^{17}	7	14	6	33	17	36	15	22	1	16	27	37	26	10	21	30	34	12	18	31	8	5	19	9	39	20	35	38	13	2	23	11	29	24	32	40	3	25	4	28
x_0^{18}	15	18	21	29	20	1	16	13	7	24	30	5	31	6	23	8	34	10	22	11	32	2	36	28	38	4	3	39	37	17	25	33	27	14	12	19	9	40	26	35
x_0^{19}	32	23	33	30	8	29	24	12	19	39	36	5	3	35	38	34	14	37	22	16	27	20	10	1	18	2	31	11	15	21	28	9	6	13	7	4	25	26	17	40
x_0^{20}	28	4	11	10	33	18	29	20	16	25	22	35	27	39	6	12	24	1	8	37	2	9	7	14	32	23	34	5	19	40	38	3	15	21	30	13	36	26	17	31
x_0^{21}	9	13	36	34	24	31	12	32	7	11	20	5	18	2	28	27	6	14	4	10	17	3	19	33	26	21	23	39	8	25	1	29	16	35	15	22	37	40	30	38
x_0^{22}	36	9	26	38	22	29	35	5	23	10	13	12	6	39	32	31	16	27	4	25	11	19	18	14	7	3	24	37	8	2	20	33	34	40	17	30	1	15	21	28
x_0^{23}	24	33	5	4	3	20	11	6	31	38	37	8	26	21	35	25	7	17	19	34	14	39	2	36	18	9	10	23	32	12	15	30	16	29	1	13	40	22	28	27
x_0^{24}	11	3	5	13	29	12	37	34	1	16	27	15	19	8	28	7	30	40	2	17	24	25	35	31	18	32	23	38	20	21	4	33	9	14	6	10	26	39	36	22
x_0^{25}	22	11	37	17	31	25	36	15	13	9	27	19	4	26	7	18	2	32	16	3	23	8	30	40	28	34	12	5	29	39	6	1	35	20	24	10	38	33	14	21
x_0^{26}	28	18	6	21	26	7	34	3	14	2	31	4	16	17	40	9	1	11	38	37	10	19	39	29	25	13	22	35	15	27	8	32	36	23	24	12	5	30	33	20
x_0^{27}	39	25	36	20	16	26	9	33	31	32	29	35	38	10	37	2	22	27	3	4	6	14	30	1	13	34	11	40	19	15	17	18	12	24	21	28	7	23	8	5
x_0^{28}	24	5	25	23	28	31	33	8	14	12	36	9	2	18	19	30	11	7	6	13	34	3	39	15	26	20	22	21	32	16	17	1	29	4	40	27	10	35	38	37
x_0^{29}	23	30	29	33	35	34	5	40	12	3	24	1	4	25	22	31	21	6	20	26	7	15	10	13	28	17	16	18	11	27	39	19	2	37	32	8	9	14	36	38
x_0^{30}	4	7	8	10	34	16	36	9	31	14	35	2	3	1	11	37	21	19	29	12	38	28	30	15	33	17	24	27	39	5	25	40	6	26	18	20	13	23	32	22
x_0^{31}	31	19	25	27	35	22	23	37	24	14	4	5	30	38	10	15	34	16	11	21	6	8	1	39	33	28	2	7	17	9	20	26	3	29	18	32	13	36	40	12
x_0^{32}	8	39	28	2	10	9	35	26	5	34	33	14	7	6	25	12	31	4	40	22	1	11	16	19	32	3	15	23	37	30	29	27	24	18	13	38	20	21	36	17
x_0^{33}	7	1	11	39	31	13	5	20	21	8	40	37	9	23	36	4	12	27	6	15	30	29	10	32	18	16	25	38	22	24	26	19	34	3	14	35	28	33	17	2
x_0^{34}	23	39	10	19	27	33	31	26	15	8	29	2	35	20	30	40	38	1	4	36	22	13	7	34	11	28	14	12	24	6	18	37	25	3	32	21	17	16	9	5
x_0^{35}	7	34	37	25	31	2	12	8	36	5	38	26	11	16	14	20	18	21	6	24	17	4	40	29	15	30	22	9	10	13	27	1	32	19	3	23	39	35	33	28
x_0^{36}	4	11	39	3	35	8	7	20	27	25	16	5	37	23	17	6	13	1	32	10	9	34	38	19	33	22	28	18	31	29	2	24	15	40	36	26	30	12	21	14
x_0^{37}	23	21	2	6	34	33	16	30	9	1	15	39	27	11	37	7	8	26	29	22	5	18	14	35	3	13	17	32	36	31	12	20	10	24	28	40	4	38	19	25
x_0^{38}	33	14	40	38	4	6	2	12	22	20	23	8	17	21	18	1	10	5	9	28	13	36	11	24	15	29	32	39	16	7	3	30	34	35	37	19	27	25	31	26

x_0^{39}	33	9	11	26	20	8	22	25	27	12	35	28	14	15	31	10	4	2	3	6	40	24	17	29	19	1	34	13	36	37	5	30	7	21	38	16	39	18	32	23
x_0^{40}	1	28	31	10	13	18	4	8	15	12	2	27	30	17	35	16	25	19	38	11	3	33	22	9	7	34	23	40	29	5	26	39	37	14	20	24	32	21	6	36
x_0^{41}	26	22	2	11	32	30	24	36	5	20	39	17	4	10	9	29	1	31	3	16	13	15	34	18	38	40	28	25	8	6	37	12	33	23	7	21	14	35	19	27
x_0^{42}	5	35	27	6	3	25	38	26	12	18	37	19	22	40	20	24	10	28	30	2	15	11	1	17	32	21	9	33	7	14	34	39	23	4	31	29	8	36	16	13
x_0^{43}	30	17	10	15	18	2	8	11	25	35	26	29	24	31	34	33	4	6	40	1	38	32	36	14	19	20	37	12	27	7	22	16	5	39	28	13	23	21	3	9
x_0^{44}	3	5	20	22	26	38	40	2	30	1	6	7	24	25	18	39	12	17	15	28	29	14	16	32	11	8	27	13	9	33	23	4	19	35	21	31	36	37	34	10
x_0^{45}	9	22	40	20	21	6	11	29	24	25	17	39	3	2	10	37	35	12	16	8	4	27	14	23	5	19	18	33	7	34	32	1	38	36	28	30	13	26	15	31
x_0^{46}	6	14	7	1	17	2	22	31	10	9	11	27	35	3	18	29	20	30	34	24	8	21	32	15	5	28	19	37	40	36	16	39	12	33	4	26	25	23	38	13
x_0^{47}	19	22	6	33	20	8	10	11	14	24	15	18	1	3	25	7	5	29	4	31	13	9	34	27	16	35	32	36	39	21	37	17	26	40	30	12	2	23	28	38
x_0^{48}	40	9	12	28	2	33	13	17	6	24	34	11	7	25	15	18	35	16	39	23	27	14	1	30	21	20	37	26	10	5	36	3	8	4	31	32	22	38	19	29
x_0^{49}	14	10	13	9	30	22	6	33	35	8	16	25	28	39	15	20	38	34	11	4	24	37	26	36	7	17	31	23	19	3	21	27	2	18	32	5	29	1	12	40
x_0^{50}	28	11	36	9	20	19	35	24	25	23	27	2	32	8	5	40	37	31	3	22	6	29	12	14	17	38	4	34	16	18	33	7	10	26	39	21	30	15	1	13
x_0^{51}	7	20	8	33	35	16	3	39	1	4	34	32	29	31	24	14	12	11	10	37	21	40	2	6	5	15	25	9	23	28	19	13	18	30	27	17	26	38	36	22
x_0^{52}	19	4	31	3	28	9	11	13	34	2	24	5	22	8	40	23	10	7	26	1	32	17	37	39	33	20	29	30	27	15	6	16	35	25	12	21	18	14	36	38
x_0^{53}	8	19	6	38	5	27	39	3	14	13	17	33	36	35	40	15	11	32	2	25	30	22	37	28	18	24	1	20	26	9	12	7	10	4	16	21	23	29	34	31
x_0^{54}	9	10	25	28	32	13	15	31	38	21	1	34	7	24	2	27	8	20	35	17	26	37	16	11	33	39	5	4	29	3	23	30	14	22	12	36	6	18	19	40
x_0^{55}	4	35	31	27	7	19	11	14	1	25	2	40	29	39	36	26	10	17	22	37	5	20	9	15	32	24	3	30	38	28	18	13	21	23	33	6	16	34	8	12
x_0^{56}	26	15	22	36	13	35	19	20	6	40	3	7	8	16	18	1	29	23	10	31	9	14	21	34	30	33	5	32	11	25	39	4	38	2	27	37	28	12	17	24
x_0^{57}	8	30	18	24	16	39	17	5	36	38	29	19	32	15	1	22	3	6	14	26	25	11	10	7	21	4	9	37	20	23	12	34	40	27	33	2	13	28	31	35
x_0^{58}	9	19	40	7	10	13	4	36	24	29	20	6	16	11	39	34	28	33	1	26	21	18	31	8	15	5	17	35	23	25	2	22	32	12	27	3	14	38	30	37
x_0^{59}	25	39	11	20	21	7	31	5	9	15	24	28	35	8	19	23	32	10	38	4	6	26	13	22	17	27	3	14	2	40	16	18	37	34	1	12	30	36	33	29
x_0^{60}	22	10	7	30	16	21	37	5	27	38	26	24	33	28	17	14	9	8	36	6	19	34	1	4	11	20	25	23	32	40	39	13	15	2	31	12	29	3	35	18
x_0^{61}	31	11	27	13	30	7	37	29	25	8	15	36	9	40	24	18	28	32	34	38	5	35	19	33	14	6	21	4	3	1	22	20	2	23	26	10	39	17	16	12
x_0^{62}	34	5	10	20	4	28	2	27	14	1	22	6	15	36	12	19	23	39	7	24	16	31	11	8	17	40	32	9	26	35	30	21	37	25	3	13	33	38	18	29
x_0^{63}	8	25	20	10	11	32	7	29	15	31	13	30	2	4	37	39	27	12	33	28	40	6	22	23	26	36	34	35	3	17	16	1	14	9	38	18	21	19	5	24
x_0^{64}	15	1	16	32	7	10	37	8	12	38	9	24	29	31	21	14	39	26	34	35	28	27	2	17	33	13	18	6	4	3	36	5	11	19	30	20	22	25	23	40
x_0^{65}	17	3	38	9	37	14	15	26	31	1	6	21	40	2	23	10	4	16	29	35	11	5	24	39	32	7	19	13	12	36	25	34	20	28	30	27	18	22	33	8
x_0^{66}	31	29	18	36	5	28	40	9	1	15	6	3	8	26	39	30	11	22	19	34	24	20	21	32	4	13	27	25	14	38	7	23	12	17	10	2	35	33	16	37
x_0^{67}	1	38	7	10	16	35	25	27	22	13	37	28	3	12	14	9	6	30	29	36	19	31	8	5	4	11	26	17	40	15	21	39	33	20	23	24	34	2	32	18
x_0^{68}	36	38	14	17	5	30	3	26	34	8	7	10	18	4	1	2	29	13	20	31	23	28	11	9	25	12	22	16	32	37	15	24	21	6	19	40	33	27	39	35
x_0^{69}	40	23	26	34	20	30	17	14	8	4	7	21	15	12	19	2	31	22	11	37	9	6	10	36	24	32	16	35	18	5	25	1	13	39	33	28	38	3	29	27
x_0^{70}	17	15	10	28	29	26	35	9	40	3	23	37	20	18	13	4	2	7	27	14	12	34	33	25	5	8	24	19	31	38	22	39	30	16	21	36	11	1	32	6
x_0^{71}	6	19	27	15	37	40	13	1	29	22	30	33	20	24	10	35	12	11	9	2	16	31	3	21	17	32	38	36	5	28	4	25	34	26	14	18	23	8	39	7
x_0^{72}	19	26	31	24	5	40	20	1	22	12	21	10	13	4	2	37	36	16	25	6	29	23	3	15	8	9	30	34	39	14	7	27	17	32	11	38	33	28	18	35
x_0^{73}	25	27	11	24	16	12	3	5	17	6	33	18	36	29	23	13	32	10	8	26	35	9	30	2	21	1	19	4	7	37	34	14	20	38	40	15	28	22	31	39
x_0^{74}	17	29	30	14	39	25	22	26	32	4	38	8	1	10	34	35	28	7	6	18	21	9	16	37	36	33	27	3	12	15	2	40	23	31	19	24	5	20	11	13
x_0^{75}	10	18	3	7	11	8	26	31	35	36	9	1	25	40	15	12	19	37	24	21	30	17	4	34	28	33	13	14	5	22	39	27	32	16	6	20	23	2	29	38
x_0^{76}	11	9	36	10	5	31	21	14	37	30	19	8	7	26	38	34	17	33	25	20	16	23	13	12	2	3	18	39	22	15	32	1	40	27	4	35	28	6	29	24
x_0^{77}	21	6	5	3	31	10	20	33	39	25	13	9	18	7	4	2	40	16	8	35	15	26	28	12	1	14	29	19	22	32	27	17	23	36	38	30	24	11	37	34
x_0^{78}	16	37	14	11	17	32	2	10	22	24	38	18	13	4	15	31	8	6	1	25	9	34	21	30	5	39	7	36	12	23	20	28	33	3	27	35	29	19	40	26
x_0^{79}	4	7	36	18	2	6	31	14	26	16	19	9	11	20	34	1	10	29	3	28	5	40	12	37	35	27	33	21	15	38	39	24	23	17	8	30	25	13	22	32
x_0^{80}	26	22	18	37	21	9	3	1	12	28	31	7	27	23	11	2	24	4	32	35	6	16	34	39	36	13	29	38	14	5	40	8	20	19	15	17	10	33	30	25

x_0^{81}	12	23	34	28	4	40	29	6	38	14	27	32	16	17	5	20	2	8	3	15	1	13	7	26	10	9	33	35	37	39	18	11	36	30	24	21	25	31	19	22
x_0^{82}	2	25	5	22	14	19	10	20	6	33	29	16	35	13	8	36	27	31	18	1	40	17	38	24	30	9	34	4	12	26	11	3	32	15	28	21	37	23	39	7
x_0^{83}	22	21	24	26	23	5	27	35	30	13	40	15	6	38	1	3	25	10	39	28	4	14	20	34	12	11	16	8	19	9	7	33	2	18	31	36	37	32	17	29
x_0^{84}	16	20	13	17	8	14	1	32	12	26	9	4	31	18	3	7	5	36	10	38	2	22	37	25	23	40	27	29	11	6	30	33	34	15	24	39	35	21	28	19
x_0^{85}	32	6	9	5	24	1	7	19	38	16	11	29	15	40	23	20	27	30	18	28	37	39	13	22	8	4	31	34	17	10	2	36	35	25	33	21	12	3	26	14
x_0^{86}	17	31	10	32	35	12	22	26	27	2	38	4	28	36	21	19	18	1	34	8	15	33	14	11	39	40	7	20	16	24	9	13	6	5	29	30	3	23	37	25
x_0^{87}	2	20	34	27	7	30	3	5	28	23	40	13	29	1	10	22	36	21	39	15	32	9	4	11	38	24	6	25	33	26	14	31	18	12	8	35	16	37	17	19
x_0^{88}	18	39	13	32	22	19	9	27	37	12	30	7	4	35	36	6	38	29	20	40	8	28	21	25	11	24	1	15	31	2	23	26	16	14	3	5	33	17	10	34
x_0^{89}	36	20	14	22	31	32	11	16	34	2	38	33	28	23	24	6	17	15	7	4	9	30	10	1	5	27	21	35	25	13	29	40	3	39	12	19	37	26	18	8
x_0^{90}	29	32	19	3	24	23	30	13	11	10	1	16	21	4	7	38	9	17	22	34	5	26	18	28	2	39	37	36	12	27	15	6	20	33	14	25	35	40	8	31
x_0^{91}	37	16	39	34	35	28	11	15	7	14	21	18	3	1	19	24	13	2	8	10	38	23	27	32	31	25	22	29	5	9	40	17	30	36	4	6	12	33	20	26
x_0^{92}	25	5	26	3	6	23	31	24	10	11	1	8	14	30	18	37	38	27	32	7	28	21	17	4	15	33	34	12	40	19	39	13	9	20	35	29	2	16	22	36
x_0^{93}	24	33	18	27	2	22	10	5	23	34	11	9	12	4	36	7	25	38	19	14	16	3	17	28	6	35	1	30	26	13	32	20	8	21	40	39	31	37	15	29
x_0^{94}	21	9	23	15	14	26	12	29	36	38	31	16	3	27	19	37	6	25	30	33	39	24	32	8	2	34	4	10	18	5	28	20	35	40	1	13	7	22	11	17
x_0^{95}	36	8	6	10	33	5	25	3	12	1	7	39	9	19	27	31	32	15	34	18	13	16	4	40	22	29	2	17	30	14	37	38	11	35	23	20	24	21	26	28
x_0^{96}	32	31	20	15	28	39	23	5	26	25	16	34	12	24	6	18	21	10	13	33	4	2	38	7	19	8	3	9	1	17	35	40	11	36	29	27	22	37	14	30
x_0^{97}	35	32	19	31	33	13	17	3	20	36	30	21	1	8	38	2	26	12	15	28	4	6	29	24	10	40	27	9	5	23	22	37	25	18	14	39	34	11	16	7
x_0^{98}	9	33	21	39	25	30	1	19	24	7	37	29	15	4	31	17	26	36	40	23	13	34	2	20	10	3	38	12	8	22	6	28	11	16	18	35	32	27	5	14
x_0^{99}	10	11	5	37	8	17	2	27	6	18	13	38	33	3	34	16	12	40	35	4	7	1	32	21	25	9	24	30	36	23	31	22	14	29	20	15	26	19	28	39
x_0^{100}	11	33	6	32	2	5	10	34	4	3	26	21	18	38	9	35	25	22	12	16	8	13	28	31	14	20	19	15	30	1	7	27	23	36	24	40	39	29	17	37

A.2 Instâncias de 40, 50 e 100 *jobs* utilizadas nos testes computacionais

As tabelas A.2.1 até à A.2.7 apresentam as instâncias utilizadas nos testes computacionais, no problema de minimização da soma dos atrasos pesados com uma só máquina (*Single Machine Total Weighted Tardiness Problem* – SMTWT). Estas instâncias foram retiradas aleatoriamente da OR-Library [http#1] e todos os *jobs* possuem um instante de chegada nulo (*ready time* - $r_i=0$). Assim, estamos perante instâncias determinísticas porque todos os parâmetros dos *jobs* são conhecidos à *priori*.

Tabela A.2.1 Instâncias 40A, 40B e 40C utilizadas nos testes computacionais

	40A				**40B**				**40C**			
Jobs	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i
1	66	1480	9	0	56	1687	1	0	73	1034	8	0
2	83	1684	8	0	25	1738	9	0	32	1260	8	0
3	4	1474	7	0	76	1663	9	0	44	1317	10	0
4	64	1438	5	0	35	1480	9	0	87	1095	1	0
5	29	1532	2	0	28	1504	5	0	67	1026	2	0
6	66	931	4	0	52	1826	1	0	63	1187	9	0
7	54	1041	6	0	21	1722	4	0	13	1138	3	0
8	6	1110	8	0	32	1660	3	0	10	1281	6	0
9	82	1219	3	0	64	1594	2	0	94	1063	1	0
10	80	1826	1	0	67	1445	8	0	11	1006	10	0
11	92	1694	7	0	48	1704	2	0	50	982	7	0
12	79	998	9	0	100	1660	3	0	90	1169	10	0
13	88	1607	5	0	94	1715	7	0	93	1051	2	0
14	52	1400	4	0	87	1701	5	0	79	1310	3	0
15	84	1174	5	0	39	1679	3	0	96	1189	10	0
16	24	1018	8	0	18	1516	5	0	39	1125	9	0
17	44	1033	4	0	78	1658	2	0	33	1097	2	0
18	60	961	10	0	80	1611	2	0	20	1212	3	0
19	75	1708	10	0	56	1502	2	0	72	1091	4	0
20	83	1405	1	0	72	1685	4	0	77	1063	8	0
21	68	1424	1	0	4	1614	9	0	11	1084	6	0
22	36	1264	7	0	70	1647	9	0	40	1336	4	0
23	88	1014	4	0	36	1689	6	0	75	1016	3	0
24	2	1314	1	0	46	1615	8	0	24	1025	4	0
25	13	1350	6	0	85	1524	9	0	52	1219	6	0
26	64	1797	3	0	31	1800	7	0	100	1186	4	0
27	25	1122	3	0	96	1654	5	0	83	990	7	0
28	29	1531	4	0	30	1752	2	0	4	1167	5	0
29	54	1424	9	0	66	1456	1	0	7	1216	4	0
30	84	1441	9	0	92	1452	6	0	12	977	7	0
31	65	1208	6	0	33	1801	4	0	49	1184	7	0
32	17	1252	6	0	18	1713	6	0	45	1155	10	0
33	99	1022	7	0	19	1761	1	0	59	1235	10	0
34	85	1410	3	0	34	1513	2	0	42	1335	5	0
35	65	938	7	0	18	1759	10	0	6	1008	9	0
36	22	1703	5	0	4	1484	10	0	75	1329	8	0
37	81	933	5	0	42	1821	6	0	16	1064	3	0
38	11	1338	9	0	94	1448	5	0	10	1242	3	0
39	62	1518	2	0	4	1666	4	0	65	1108	1	0
40	100	1090	7	0	89	1611	3	0	29	1328	1	0

Tabela A.2.2 Instâncias 40D, 40E e 40F utilizadas nos testes computacionais

	40D				**40E**				**40F**			
Jobs	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i
1	22	1192	7	0	7	1471	5	0	76	1145	1	0
2	83	1284	7	0	70	1794	10	0	54	1091	7	0
3	90	1086	10	0	52	1514	1	0	14	1222	6	0
4	4	1269	1	0	52	1473	2	0	32	1290	3	0
5	33	1008	5	0	86	1702	2	0	100	1047	10	0
6	16	1002	7	0	66	1583	8	0	37	1011	7	0
7	21	1202	1	0	70	1640	8	0	69	1058	2	0
8	17	943	8	0	60	1683	5	0	36	1313	8	0
9	34	1153	10	0	65	1491	6	0	27	1351	1	0
10	54	960	1	0	70	1519	4	0	7	1076	8	0
11	51	1201	6	0	27	1702	8	0	39	1163	8	0
12	33	1204	7	0	41	1527	4	0	52	1004	10	0
13	88	1074	6	0	42	1701	1	0	74	1141	4	0
14	93	983	1	0	88	1675	1	0	67	1061	3	0
15	94	1115	6	0	21	1421	6	0	93	1259	1	0
16	13	1066	8	0	15	1718	5	0	49	1306	10	0
17	85	1034	6	0	40	1639	10	0	89	985	10	0
18	84	981	2	0	80	1742	7	0	73	1173	8	0
19	41	1199	4	0	28	1749	2	0	79	1154	2	0
20	21	1174	1	0	7	1653	7	0	98	1325	10	0
21	43	965	1	0	13	1441	5	0	45	1352	6	0
22	30	1222	6	0	58	1722	8	0	36	1169	7	0
23	64	1242	6	0	4	1691	7	0	24	1189	3	0
24	25	1020	6	0	43	1612	7	0	71	1241	3	0
25	10	993	10	0	41	1424	6	0	47	1210	8	0
26	89	1298	10	0	89	1615	2	0	19	1209	10	0
27	10	1186	2	0	80	1809	6	0	32	1331	1	0
28	96	1148	6	0	54	1533	6	0	47	1128	2	0
29	59	1032	2	0	34	1648	1	0	25	1102	3	0
30	70	1132	7	0	92	1702	8	0	37	983	2	0
31	5	1182	6	0	66	1450	8	0	86	1232	8	0
32	8	1115	3	0	72	1614	5	0	5	1131	2	0
33	93	1299	3	0	29	1675	7	0	37	1358	3	0
34	37	1062	9	0	40	1435	6	0	60	1339	5	0
35	54	1094	5	0	53	1441	6	0	46	1260	6	0
36	37	949	1	0	35	1485	4	0	28	1074	9	0
37	44	1175	8	0	91	1746	8	0	72	1283	10	0
38	4	993	9	0	58	1732	6	0	11	1068	6	0
39	39	1293	9	0	6	1727	5	0	52	1008	4	0
40	65	959	5	0	82	1612	7	0	4	1023	9	0

Tabela A.2.3 Instâncias 50A, 50B e 50C utilizadas nos testes computacionais

	50A				**50B**				**50C**			
Jobs	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i
1	49	2455	3	0	78	2098	4	0	38	2001	4	0
2	64	2462	7	0	92	2329	1	0	99	2105	7	0
3	78	2372	9	0	85	2563	9	0	95	2199	4	0
4	33	2082	4	0	47	2376	3	0	36	2166	9	0
5	56	2223	7	0	60	1894	9	0	97	2185	6	0
6	58	2166	6	0	3	2698	7	0	83	1766	8	0
7	52	2356	5	0	48	2610	5	0	57	2202	4	0
8	82	2382	8	0	89	2515	10	0	3	2134	7	0
9	81	2265	7	0	56	2780	2	0	68	2107	10	0
10	44	2099	3	0	84	2188	4	0	57	1824	10	0
11	57	2045	8	0	83	2239	2	0	8	1900	1	0
12	49	2283	5	0	65	1889	4	0	39	2060	3	0
13	99	2379	4	0	97	2174	7	0	91	2100	7	0
14	100	2168	2	0	12	2085	5	0	47	1925	3	0
15	79	2394	1	0	65	1743	4	0	70	1912	9	0
16	79	2007	3	0	60	2396	1	0	17	2102	2	0
17	28	2222	3	0	30	2695	1	0	92	1793	2	0
18	70	2473	10	0	4	2695	3	0	95	2156	4	0
19	91	2390	7	0	39	2245	10	0	2	1877	5	0
20	38	2034	3	0	50	1874	10	0	72	2042	8	0
21	27	2437	7	0	20	1889	7	0	92	1871	9	0
22	61	2259	3	0	19	2617	3	0	57	1850	2	0
23	61	1978	1	0	13	2388	3	0	7	1847	7	0
24	4	2462	3	0	100	2006	9	0	8	1960	6	0
25	93	2172	6	0	45	2569	10	0	99	2096	9	0
26	48	2459	7	0	54	1911	4	0	7	2124	2	0
27	44	2493	10	0	61	2269	1	0	28	1925	6	0
28	71	2091	3	0	66	2298	3	0	92	2150	8	0
29	65	2424	9	0	99	2546	2	0	14	2073	10	0
30	72	2417	6	0	48	2479	3	0	32	2184	4	0
31	87	2385	5	0	23	2367	7	0	46	2154	5	0
32	50	2212	1	0	95	2409	5	0	57	1799	8	0
33	24	2458	7	0	94	2742	10	0	27	1761	9	0
34	37	2441	5	0	83	2320	5	0	37	1864	3	0
35	28	2083	6	0	5	1895	6	0	85	2164	9	0
36	67	2366	7	0	26	1702	4	0	4	1904	9	0
37	90	2038	6	0	55	2484	5	0	20	2241	6	0
38	14	2184	8	0	77	2605	3	0	41	1929	6	0
39	94	2488	10	0	78	2188	4	0	10	2055	6	0
40	64	2475	6	0	85	2688	6	0	98	1945	4	0
41	23	2271	1	0	22	2055	1	0	64	2232	3	0
42	42	2090	1	0	93	1917	4	0	79	2118	9	0
43	6	2082	10	0	78	1715	4	0	25	2044	4	0
44	96	2475	10	0	93	2772	5	0	68	1809	4	0
45	26	2309	2	0	55	2443	10	0	23	1858	5	0
46	22	2331	6	0	31	2223	6	0	4	1945	9	0
47	72	2073	7	0	9	1694	1	0	77	1852	7	0
48	38	2235	2	0	58	2164	3	0	11	1958	2	0
49	41	2406	9	0	25	2314	6	0	49	2159	7	0
50	21	2032	8	0	34	2000	9	0	80	2241	9	0

Tabela A.2.4 Instâncias 50D, 50E e 50F utilizadas nos testes computacionais

	50D				**50E**				**50F**			
Jobs	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i	p_i	d_i	w_i	r_i
1	18	2088	2	0	98	1617	4	0	39	1666	8	0
2	22	2044	4	0	71	2304	9	0	72	2095	10	0
3	34	2261	8	0	30	2146	2	0	93	1110	4	0
4	28	2132	9	0	40	1747	3	0	82	1202	4	0
5	83	2112	6	0	65	1925	2	0	64	2095	1	0
6	79	2348	6	0	85	2082	9	0	82	1108	2	0
7	95	2301	5	0	32	2073	1	0	93	1903	9	0
8	48	2195	6	0	14	1981	10	0	64	1411	3	0
9	40	2318	10	0	55	1887	2	0	15	1606	8	0
10	60	2265	8	0	50	2101	3	0	62	1432	10	0
11	9	1934	2	0	50	2451	1	0	20	1135	5	0
12	91	2228	5	0	59	2465	2	0	29	2002	10	0
13	41	2208	5	0	86	2463	8	0	44	1113	3	0
14	25	2025	6	0	24	2017	3	0	65	1451	1	0
15	100	1948	7	0	84	2115	10	0	34	1675	9	0
16	52	1944	1	0	78	1817	3	0	97	1171	5	0
17	4	2324	9	0	69	2317	9	0	42	1788	1	0
18	24	2373	3	0	79	1754	4	0	7	1528	4	0
19	5	2208	6	0	77	2074	10	0	85	2077	2	0
20	61	2100	3	0	39	2286	7	0	8	1342	1	0
21	14	2263	10	0	3	2366	5	0	43	1901	1	0
22	99	2354	10	0	3	2088	5	0	74	1974	8	0
23	79	1949	8	0	11	2283	2	0	15	1133	6	0
24	94	2140	10	0	76	2080	7	0	17	1538	9	0
25	19	2160	6	0	79	1796	1	0	20	1646	2	0
26	79	2163	4	0	100	1749	10	0	15	1553	10	0
27	92	1911	7	0	10	1920	3	0	53	1913	8	0
28	64	2388	6	0	87	2223	6	0	92	2013	2	0
29	34	2307	5	0	79	1604	10	0	87	2111	8	0
30	88	1993	10	0	91	2013	8	0	44	1834	9	0
31	28	1972	2	0	39	2178	9	0	24	1401	4	0
32	25	2219	4	0	39	1889	7	0	18	1955	10	0
33	40	1965	4	0	74	1624	10	0	99	1581	1	0
34	37	2218	10	0	10	1792	3	0	41	1443	1	0
35	89	2217	6	0	2	1539	1	0	38	1986	3	0
36	74	2254	1	0	61	2255	2	0	97	1979	5	0
37	83	2185	9	0	27	2057	9	0	52	1961	10	0
38	31	2246	3	0	8	2234	4	0	54	1482	3	0
39	91	2166	7	0	44	1560	10	0	99	1106	7	0
40	47	2000	7	0	50	1834	2	0	45	1835	5	0
41	74	1959	9	0	91	2507	2	0	81	1892	4	0
42	62	2300	5	0	52	1809	4	0	94	1902	2	0
43	65	2385	8	0	52	2149	10	0	1	1671	1	0
44	19	2041	2	0	68	2330	10	0	95	1835	4	0
45	93	2097	8	0	10	2506	9	0	63	2102	2	0
46	16	2270	10	0	24	1910	10	0	39	1603	5	0
47	61	2101	6	0	52	2347	6	0	29	1706	8	0
48	9	2216	9	0	42	1972	6	0	94	1623	9	0
49	80	1914	3	0	19	1549	8	0	16	1716	9	0
50	67	1892	7	0	21	1943	5	0	31	1200	7	0

Tabela A.2.5 Instâncias 100A e 100B utilizadas nos testes computacionais

100A										100B									
Jobs	p_i	d_i	w_i	r_i	**Jobs**	p_i	d_i	w_i	r_i	***Jobs***	p_i	d_i	w_i	r_i	**Jobs**	p_i	d_i	w_i	r_i
1	1	3907	10	0	51	94	3752	10	0	1	2	4243	3	0	51	41	4813	4	0
2	5	4261	8	0	52	58	4122	6	0	2	12	3678	8	0	52	83	5209	8	0
3	5	4520	5	0	53	99	4227	10	0	3	14	3643	7	0	53	32	3392	3	0
4	13	4567	10	0	54	90	4536	9	0	4	22	5089	10	0	54	88	3439	8	0
5	16	3803	10	0	55	50	4204	5	0	5	19	5193	7	0	55	91	5271	8	0
6	15	3814	9	0	56	91	3713	9	0	6	20	4692	7	0	56	48	5446	4	0
7	14	4248	8	0	57	61	4113	6	0	7	29	4759	10	0	57	84	4656	7	0
8	14	3875	8	0	58	82	4240	8	0	8	18	4115	6	0	58	73	4738	6	0
9	18	4015	9	0	59	72	4753	7	0	9	28	4844	9	0	59	98	5641	8	0
10	22	4062	10	0	60	31	4095	3	0	10	32	4507	10	0	60	62	4249	5	0
11	22	4187	10	0	61	73	4068	7	0	11	20	5031	6	0	61	25	4808	2	0
12	20	4393	9	0	62	92	3815	8	0	12	29	4322	8	0	62	50	5512	4	0
13	20	4293	8	0	63	53	4537	4	0	13	34	4773	9	0	63	77	4137	6	0
14	13	3717	5	0	64	53	4762	4	0	14	35	4220	9	0	64	91	4129	7	0
15	8	4727	3	0	65	94	3726	7	0	15	8	5624	2	0	65	53	4116	4	0
16	14	3797	5	0	66	81	4384	6	0	16	40	4807	10	0	66	53	4897	4	0
17	31	3857	10	0	67	55	4281	4	0	17	41	4637	10	0	67	67	5186	5	0
18	23	4015	7	0	68	28	4142	2	0	18	39	3776	9	0	68	72	5425	5	0
19	10	4508	3	0	69	87	4582	6	0	19	39	4376	9	0	69	87	4781	6	0
20	21	3738	6	0	70	89	4546	6	0	20	18	5585	4	0	70	78	4569	5	0
21	19	3767	5	0	71	89	4471	6	0	21	47	3899	10	0	71	47	5333	3	0
22	36	3857	9	0	72	60	3811	4	0	22	48	5443	10	0	72	79	5112	5	0
23	14	3727	3	0	73	60	4151	4	0	23	10	3917	2	0	73	100	4848	6	0
24	24	4558	5	0	74	79	4681	5	0	24	10	3947	2	0	74	34	3814	2	0
25	39	3782	8	0	75	95	3724	6	0	25	47	3526	9	0	75	54	4386	3	0
26	46	4202	9	0	76	32	4235	2	0	26	44	4691	8	0	76	93	4272	5	0
27	46	4547	9	0	77	80	4768	5	0	27	59	5244	10	0	77	77	4972	4	0
28	37	3974	7	0	78	50	4339	3	0	28	61	5230	10	0	78	98	4009	5	0
29	27	4612	5	0	79	100	4025	6	0	29	57	5381	9	0	79	59	4701	3	0
30	49	4360	9	0	80	51	4707	3	0	30	52	4579	8	0	80	79	3949	4	0
31	51	4760	9	0	81	18	3858	1	0	31	40	5519	6	0	81	85	4053	4	0
32	49	4318	8	0	82	72	3713	4	0	32	48	5433	7	0	82	44	4136	2	0
33	25	4562	4	0	83	55	4761	3	0	33	49	3555	7	0	83	68	4990	3	0
34	39	4066	6	0	84	74	3743	4	0	34	49	4440	7	0	84	91	3872	4	0
35	33	4281	5	0	85	97	4126	5	0	35	66	3908	9	0	85	96	4811	4	0
36	61	4502	9	0	86	97	4022	5	0	36	24	5337	3	0	86	50	3748	2	0
37	35	4740	5	0	87	79	4671	4	0	37	74	4146	9	0	87	76	5039	3	0
38	65	4188	9	0	88	88	4591	4	0	38	74	3430	9	0	88	79	3685	3	0
39	22	4324	3	0	89	74	3735	3	0	39	42	4220	5	0	89	82	5580	3	0
40	68	3819	9	0	90	54	4761	2	0	40	44	5414	5	0	90	66	4340	2	0
41	69	4642	9	0	91	99	4648	3	0	41	81	4898	9	0	91	70	4843	2	0
42	41	4243	5	0	92	73	4507	2	0	42	91	5233	10	0	92	79	4620	2	0
43	17	4378	2	0	93	87	4361	2	0	43	46	4145	5	0	93	90	4134	2	0
44	69	4207	8	0	94	47	4262	1	0	44	66	5546	7	0	94	50	4278	1	0
45	78	4527	9	0	95	100	4231	2	0	45	58	4826	6	0	95	59	4741	1	0
46	53	4609	6	0	96	52	4412	1	0	46	29	5392	3	0	96	59	4378	1	0
47	72	3928	8	0	97	65	3888	1	0	47	87	5123	9	0	97	62	4479	1	0
48	54	3729	6	0	98	67	3864	1	0	48	68	4819	7	0	98	78	4888	1	0
49	91	3790	10	0	99	86	4288	1	0	49	50	4515	5	0	99	83	4495	1	0
50	65	4366	7	0	100	88	3722	1	0	50	90	5124	9	0	100	92	4710	1	0

Tabela A.2.6 Instâncias 100C e 100D utilizadas nos testes computacionais

100C										100D									
Jobs	p_i	d_i	w_i	r_i	**Jobs**	p_i	d_i	w_i	r_i	***Jobs***	p_i	d_i	w_i	r_i	**Jobs**	p_i	d_i	w_i	r_i
1	1	3746	10	0	51	38	3873	4	0	1	1	4151	4	0	51	89	3498	10	0
2	1	3957	7	0	52	58	4245	6	0	2	2	3919	7	0	52	75	3965	8	0
3	3	3397	10	0	53	39	3446	4	0	3	2	3858	5	0	53	58	3516	6	0
4	3	3642	7	0	54	90	3906	9	0	4	4	4171	7	0	54	97	3940	10	0
5	7	4188	10	0	55	90	3714	9	0	5	3	4016	4	0	55	39	3730	4	0
6	5	4244	6	0	56	72	4062	7	0	6	7	3980	8	0	56	98	3417	10	0
7	9	3440	10	0	57	21	4129	2	0	7	3	3720	3	0	57	89	4290	9	0
8	6	3949	6	0	58	85	4053	8	0	8	13	3656	10	0	58	41	4244	4	0
9	12	4035	9	0	59	89	3527	8	0	9	4	4052	3	0	59	93	3438	9	0
10	15	4280	8	0	60	80	4302	7	0	10	4	4207	3	0	60	98	4045	9	0
11	12	4075	6	0	61	24	3798	2	0	11	6	4232	4	0	61	44	4231	4	0
12	6	3529	3	0	62	85	3907	7	0	12	7	3883	4	0	62	88	3641	8	0
13	19	3813	9	0	63	51	3782	4	0	13	2	3515	1	0	63	66	4109	6	0
14	22	4145	10	0	64	91	3806	7	0	14	13	3989	6	0	64	80	3412	7	0
15	17	4256	6	0	65	13	4174	1	0	15	11	4151	5	0	65	74	3960	6	0
16	26	4176	9	0	66	26	4048	2	0	16	18	4171	7	0	66	26	3638	2	0
17	24	4123	8	0	67	92	3985	7	0	17	11	3901	4	0	67	67	4128	5	0
18	30	3564	9	0	68	93	3437	7	0	18	3	4012	1	0	68	57	3716	4	0
19	21	4025	6	0	69	82	3854	6	0	19	30	3542	9	0	69	43	3461	3	0
20	23	3623	6	0	70	84	3530	6	0	20	22	3821	6	0	70	15	4270	1	0
21	27	4217	7	0	71	29	4157	2	0	21	39	3649	10	0	71	61	4020	4	0
22	25	3466	6	0	72	61	3739	4	0	22	8	3719	2	0	72	93	3653	6	0
23	25	3802	6	0	73	77	3561	5	0	23	40	4069	9	0	73	97	4247	6	0
24	42	3673	10	0	74	31	3761	2	0	24	28	3645	6	0	74	17	3438	1	0
25	34	4000	8	0	75	96	3469	6	0	25	33	3689	7	0	75	52	4279	3	0
26	30	3991	7	0	76	54	3753	3	0	26	19	4245	4	0	76	90	3967	5	0
27	40	3522	9	0	77	76	3712	4	0	27	49	4266	10	0	77	92	3832	5	0
28	45	3719	9	0	78	76	3405	4	0	28	35	3610	7	0	78	74	3377	4	0
29	53	4049	10	0	79	60	3758	3	0	29	25	3867	5	0	79	93	3501	5	0
30	53	3608	10	0	80	40	4191	2	0	30	44	4229	8	0	80	56	3907	3	0
31	54	4240	10	0	81	66	3514	3	0	31	17	3833	3	0	81	96	4233	5	0
32	39	4050	7	0	82	70	3843	3	0	32	40	3870	7	0	82	58	3385	3	0
33	46	3436	8	0	83	79	3940	3	0	33	52	3458	9	0	83	62	3562	3	0
34	12	4162	2	0	84	27	3909	1	0	34	58	4232	10	0	84	69	3594	3	0
35	18	3510	3	0	85	90	3626	3	0	35	35	3938	6	0	85	93	4167	4	0
36	49	3932	8	0	86	62	3717	2	0	36	6	3938	1	0	86	71	3581	3	0
37	65	3468	10	0	87	95	3513	3	0	37	45	3907	7	0	87	48	3585	2	0
38	66	3439	10	0	88	32	3642	1	0	38	13	3803	2	0	88	75	3351	3	0
39	42	3528	6	0	89	32	3765	1	0	39	67	3938	10	0	89	75	3735	3	0
40	56	3764	8	0	90	68	3586	2	0	40	47	3612	7	0	90	26	3410	1	0
41	64	3603	9	0	91	40	4281	1	0	41	57	3390	8	0	91	56	3550	2	0
42	24	3920	3	0	92	41	4153	1	0	42	46	3596	6	0	92	64	4136	2	0
43	64	3908	8	0	93	86	4107	2	0	43	69	3487	9	0	93	98	4041	3	0
44	40	4173	5	0	94	96	4265	2	0	44	47	4031	6	0	94	99	3362	3	0
45	65	4195	8	0	95	67	4262	1	0	45	32	4065	4	0	95	35	4179	1	0
46	17	3788	2	0	96	80	3987	1	0	46	16	4130	2	0	96	78	3547	2	0
47	34	4129	4	0	97	80	3501	1	0	47	66	3647	8	0	97	47	4050	1	0
48	26	3436	3	0	98	81	3482	1	0	48	50	3370	6	0	98	63	4249	1	0
49	91	4280	10	0	99	85	3923	1	0	49	67	4088	8	0	99	71	3611	1	0
50	66	4295	7	0	100	90	3609	1	0	50	35	3514	4	0	100	76	3811	1	0

Tabela A.2.7 Instâncias 100E e 100F utilizadas nos testes computacionais

100E										100F									
Jobs	p_i	d_i	w_i	r_i	**Jobs**	p_i	d_i	w_i	r_i	***Jobs***	p_i	d_i	w_i	r_i	**Jobs**	p_i	d_i	w_i	r_i
1	2	3363	10	0	51	37	3985	4	0	1	3	5066	10	0	51	84	3797	9	0
2	2	4154	10	0	52	84	3889	9	0	2	3	4524	10	0	52	19	3891	2	0
3	3	4209	10	0	53	56	3393	6	0	3	4	4905	10	0	53	48	4691	5	0
4	8	3984	8	0	54	86	3685	9	0	4	3	3711	7	0	54	48	4025	5	0
5	7	3946	7	0	55	100	3696	10	0	5	9	3326	10	0	55	20	4093	2	0
6	8	4095	8	0	56	71	4173	7	0	6	12	3400	10	0	56	21	4537	2	0
7	16	3941	10	0	57	62	3377	6	0	7	6	3512	5	0	57	95	3688	9	0
8	10	3704	5	0	58	66	4054	6	0	8	9	4266	6	0	58	97	4784	9	0
9	13	3429	6	0	59	70	4280	6	0	9	13	5065	8	0	59	76	3704	7	0
10	18	3843	8	0	60	35	4211	3	0	10	7	4695	3	0	60	76	3328	7	0
11	7	4106	3	0	61	48	4135	4	0	11	20	5234	8	0	61	90	4043	8	0
12	24	3749	10	0	62	49	3803	4	0	12	20	5015	8	0	62	68	3980	6	0
13	24	4101	9	0	63	13	3534	1	0	13	29	4578	9	0	63	71	3288	6	0
14	16	4284	6	0	64	65	4284	5	0	14	20	3668	6	0	64	95	3708	8	0
15	19	3577	7	0	65	92	3594	7	0	15	10	4987	3	0	65	74	4532	6	0
16	20	4042	7	0	66	92	3764	7	0	16	36	4324	10	0	66	99	3201	8	0
17	20	3993	6	0	67	80	3864	6	0	17	29	3997	8	0	67	26	4013	2	0
18	29	3801	7	0	68	27	3634	2	0	18	12	3956	3	0	68	93	4329	7	0
19	42	4097	10	0	69	84	3907	6	0	19	32	5284	8	0	69	93	3485	7	0
20	17	4056	4	0	70	85	3438	6	0	20	16	4577	4	0	70	93	3662	7	0
21	30	3822	7	0	71	57	3731	4	0	21	37	3860	9	0	71	40	3507	3	0
22	44	3608	10	0	72	33	4129	2	0	22	22	3430	5	0	72	54	3531	4	0
23	14	3509	3	0	73	51	3871	3	0	23	41	4816	9	0	73	81	4611	6	0
24	24	3565	5	0	74	70	3430	4	0	24	30	3596	6	0	74	95	4690	7	0
25	24	4158	5	0	75	55	4008	3	0	25	10	4591	2	0	75	72	3967	5	0
26	48	3752	10	0	76	93	3394	5	0	26	20	3384	4	0	76	58	4655	4	0
27	24	3620	5	0	77	57	3469	3	0	27	16	3734	3	0	77	61	4207	4	0
28	39	3432	8	0	78	57	3558	3	0	28	16	4581	3	0	78	79	5285	5	0
29	44	3920	9	0	79	19	3665	1	0	29	38	3640	7	0	79	64	4420	4	0
30	32	3662	6	0	80	83	3481	4	0	30	33	4526	6	0	80	99	4363	5	0
31	16	3537	3	0	81	42	3978	2	0	31	33	3821	6	0	81	83	4615	4	0
32	11	3692	2	0	82	22	3860	1	0	32	23	3462	4	0	82	93	4022	4	0
33	56	3726	9	0	83	46	3641	2	0	33	47	3853	8	0	83	97	4097	4	0
34	65	4061	10	0	84	24	3875	1	0	34	53	5238	9	0	84	98	3319	4	0
35	13	4115	2	0	85	51	4260	2	0	35	19	3396	3	0	85	74	3705	3	0
36	67	3845	10	0	86	77	3872	3	0	36	45	4747	7	0	86	100	4936	4	0
37	54	3718	8	0	87	52	3628	2	0	37	65	4469	10	0	87	53	3956	2	0
38	34	3540	5	0	88	95	3877	3	0	38	34	3672	5	0	88	56	3825	2	0
39	50	4165	7	0	89	66	4260	2	0	39	58	4093	8	0	89	90	5225	3	0
40	75	3548	10	0	90	35	3475	1	0	40	73	4077	10	0	90	67	4893	2	0
41	41	3442	5	0	91	71	4091	2	0	41	75	3253	10	0	91	72	5086	2	0
42	76	3421	9	0	92	71	4016	2	0	42	76	4300	10	0	92	75	4349	2	0
43	86	3379	10	0	93	41	3786	1	0	43	46	3483	6	0	93	82	3193	2	0
44	88	3965	10	0	94	41	3515	1	0	44	62	4464	8	0	94	44	3558	1	0
45	53	4158	6	0	95	83	3907	2	0	45	47	4654	6	0	95	96	3456	2	0
46	62	3954	7	0	96	63	3457	1	0	46	72	4918	9	0	96	49	4905	1	0
47	73	4007	8	0	97	66	3973	1	0	47	81	3682	10	0	97	68	4378	1	0
48	55	4250	6	0	98	68	3744	1	0	48	67	3797	8	0	98	76	5175	1	0
49	83	3379	9	0	99	68	3648	1	0	49	76	5151	9	0	99	80	3469	1	0
50	74	3870	8	0	100	73	3786	1	0	50	60	3508	7	0	100	85	4797	1	0

Apêndice B

Instâncias utilizadas nos problemas de sequenciamento *Job-Shop*

B.1 Instâncias dos problemas de sequenciamento *Job-Shop* utilizadas nos testes computacionais

As tabelas a seguir apresentam as instâncias dos problemas de sequenciamento *Job-Shop* utilizadas nos testes computacionais, retiradas da OR-Library [http#1]. Cada linha contém a sequência das operações nas máquinas para um *job* e os respectivos tempos de processamento requeridos nas máquinas.

Tabela B.1 Instância FT06

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	3	1	1	3	2	6	4	7	6	3	5	6
2	2	8	3	5	5	10	6	10	1	10	4	4
3	3	5	4	4	6	8	1	9	2	1	5	7
4	2	5	1	5	3	5	4	3	5	8	6	9
5	3	9	2	3	5	5	6	4	1	3	4	1
6	2	3	4	3	6	9	1	10	5	4	3	1

Tabela B.2 Instância FT10

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	1	29	2	78	3	9	4	36	5	49	6	11	7	62	8	56	9	44	10	21
2	1	43	3	90	5	75	10	11	4	69	2	28	7	46	6	46	8	72	9	30
3	2	91	1	85	4	39	3	74	9	90	6	10	8	12	7	89	10	45	5	33
4	2	81	3	95	1	71	5	99	7	9	9	52	8	85	4	98	10	22	6	43
5	3	14	1	6	2	22	6	61	4	26	5	69	9	21	8	49	10	72	7	53
6	3	84	2	2	6	52	4	95	9	48	10	72	1	47	7	65	5	6	8	25
7	2	46	1	37	4	61	3	13	7	32	6	21	10	32	9	89	8	30	5	55
8	3	31	1	86	2	46	6	74	5	32	7	88	9	19	10	48	8	36	4	79
9	1	76	2	69	4	76	6	51	3	85	10	11	7	40	8	89	5	26	9	74
10	2	85	1	13	3	61	7	7	9	64	10	76	6	47	4	52	5	90	8	45

Tabela B.3 Instância FT20

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	1	29	2	9	3	49	4	62	5	44
2	1	43	2	75	4	69	3	46	5	72
3	2	91	1	39	3	90	5	12	4	45
4	2	81	1	71	5	9	3	85	4	22
5	3	14	2	22	1	26	4	21	5	72
6	3	84	2	52	5	48	1	47	4	6
7	2	46	1	61	3	32	4	32	5	30
8	3	31	2	46	1	32	4	19	5	36
9	1	76	4	76	3	85	2	40	5	26
10	2	85	3	61	1	64	4	47	5	90
11	2	78	4	36	1	11	5	56	3	21
12	3	90	1	11	2	28	4	46	5	30
13	1	85	3	74	2	10	4	89	5	33
14	3	95	1	99	2	52	4	98	5	43
15	1	6	2	61	5	69	3	49	4	53
16	2	2	1	95	4	72	5	65	3	25
17	1	37	3	13	2	21	4	89	5	55
18	1	86	2	74	5	88	3	48	4	79
19	2	69	3	51	1	11	4	89	5	74
20	1	13	2	7	3	76	4	52	5	45

Tabela B.4 Instância LA01

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	2	21	1	53	5	95	4	55	3	34
2	1	21	4	52	5	16	3	26	2	71
3	4	39	5	98	2	42	3	31	1	12
4	2	77	1	55	5	79	3	66	4	77
5	1	83	4	34	3	64	2	19	5	37
6	2	54	3	43	5	79	1	92	4	62
7	4	69	5	77	2	87	3	87	1	93
8	3	38	1	60	2	41	4	24	5	83
9	4	17	2	49	5	25	1	44	3	98
10	5	77	4	79	3	43	2	75	1	96

Tabela B.5 Instância LA06

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	2	21	3	34	5	95	1	53	4	55
2	4	52	5	16	2	71	3	26	1	21
3	3	31	1	12	2	42	4	39	5	98
4	4	77	2	77	5	79	1	55	3	66
5	5	37	4	34	3	64	2	19	1	83
6	3	43	2	54	1	92	4	62	5	79
7	1	93	4	69	2	87	5	77	3	87
8	1	60	2	41	3	38	5	83	4	24
9	3	98	4	17	5	25	1	44	2	49
10	1	96	5	77	4	79	2	75	3	43
11	5	28	3	35	1	95	4	76	2	7
12	1	61	5	10	3	95	2	9	4	35
13	5	59	4	16	2	91	3	59	1	46
14	5	43	2	52	1	28	3	27	4	50
15	1	87	2	45	3	39	5	9	4	41

Tabela B.6 Instância LA11

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	3	34	2	21	1	53	4	55	5	95
2	1	21	4	52	2	71	5	16	3	26
3	1	12	2	42	3	31	5	98	4	39
4	3	66	4	77	5	79	1	55	2	77
5	1	83	5	37	4	34	2	19	3	64
6	5	79	3	43	1	92	4	62	2	54
7	1	93	5	77	3	87	2	87	4	69
8	5	83	4	24	2	41	3	38	1	60
9	5	25	2	49	1	44	3	98	4	17
10	1	96	2	75	3	43	5	77	4	79
11	1	95	4	76	2	7	5	28	3	35
12	5	10	3	95	1	61	2	9	4	35
13	2	91	3	59	5	59	1	46	4	16
14	3	27	2	52	5	43	1	28	4	50
15	5	9	1	87	4	41	3	39	2	45
16	2	54	1	20	5	43	4	14	3	71
17	5	33	2	28	4	26	1	78	3	37
18	2	89	1	33	3	8	4	66	5	42
19	5	84	1	69	3	94	2	74	4	27
20	5	81	3	45	2	78	4	69	1	96

Tabela B.7 Instância LA16

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	2	21	7	71	10	16	9	52	8	26	3	34	1	53	5	21	4	55	6	95
2	5	55	3	31	6	98	10	79	1	12	8	66	2	42	9	77	7	77	4	39
3	4	34	3	64	9	62	2	19	5	92	10	79	8	43	7	54	1	83	6	37
4	2	87	4	69	3	87	8	38	9	24	10	83	7	41	1	93	6	77	5	60
5	3	98	1	44	6	25	7	75	8	43	2	49	5	96	10	77	4	17	9	79
6	3	35	4	76	6	28	10	10	5	61	7	9	1	95	9	35	2	7	8	95
7	4	16	3	59	1	46	2	91	10	43	9	50	7	52	6	59	5	28	8	27
8	2	45	1	87	4	41	5	20	7	54	10	43	9	14	6	9	3	39	8	71
9	5	33	3	37	9	66	6	33	4	26	8	8	2	28	7	89	10	42	1	78
10	9	69	10	81	3	94	5	96	4	27	1	69	8	45	7	78	2	74	6	84

Tabela B.8 Instância LA21

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	3	34	4	55	6	95	10	16	5	21	7	71	1	53	9	52	2	21	8	26
2	4	39	3	31	1	12	2	42	10	79	9	77	7	77	6	98	5	55	8	66
3	2	19	1	83	4	34	5	92	7	54	10	79	9	62	6	37	3	64	8	43
4	5	60	3	87	9	24	6	77	4	69	8	38	2	87	7	41	10	83	1	93
5	9	79	10	77	3	98	5	96	4	17	1	44	8	43	7	75	2	49	6	25
6	9	35	8	95	7	9	10	10	3	35	2	7	6	28	5	61	1	95	4	76
7	5	28	6	59	4	16	10	43	1	46	9	50	7	52	8	27	3	59	2	91
8	6	9	5	20	3	39	7	54	2	45	8	71	1	87	4	41	10	43	9	14
9	2	28	6	33	1	78	4	26	3	37	8	8	9	66	7	89	10	42	5	33
10	3	94	6	84	7	78	10	81	2	74	4	27	9	69	1	69	8	45	5	96
11	2	31	5	24	1	20	3	17	10	25	9	81	6	76	4	87	8	32	7	18
12	6	28	10	97	1	58	5	45	7	76	4	99	3	23	2	72	9	90	8	86
13	6	27	10	48	9	27	8	62	5	98	7	67	4	48	1	42	2	46	3	17
14	2	12	9	50	1	80	3	50	10	80	4	19	6	28	7	63	5	94	8	98
15	5	61	4	55	7	37	6	14	3	50	9	79	2	41	10	72	8	18	1	75

Tabela B.9 Instância LA26

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	9	52	8	26	7	71	10	16	3	34	2	21	6	95	5	21	1	53	4	55
2	5	55	6	98	4	39	10	79	1	12	9	77	7	77	8	66	3	31	2	42
3	6	37	5	92	3	64	7	54	2	19	8	43	1	83	4	34	10	79	9	62
4	2	87	6	77	1	93	4	69	3	87	8	38	9	24	7	41	10	83	5	60
5	3	98	6	25	7	75	10	77	2	49	4	17	9	79	1	44	8	43	5	96
6	2	7	5	61	1	95	3	35	10	10	9	35	6	28	4	76	8	95	7	9
7	6	59	10	43	1	46	5	28	7	52	4	16	3	59	2	91	9	50	8	27
8	6	9	10	43	9	14	8	71	5	20	7	54	4	41	1	87	2	45	3	39
9	2	28	9	66	1	78	3	37	10	42	4	26	6	33	7	89	5	33	8	8
10	5	96	4	27	7	78	6	84	3	94	9	69	2	74	10	81	8	45	1	69
11	5	24	8	32	10	25	3	17	4	87	9	81	6	76	7	18	2	31	1	20
12	9	90	6	28	2	72	8	86	3	23	4	99	7	76	10	97	5	45	1	58
13	3	17	5	98	4	48	2	46	9	27	7	67	8	62	1	42	10	48	6	27
14	1	80	9	50	4	19	8	98	6	28	3	50	5	94	7	63	2	12	10	80
15	10	72	1	75	5	61	9	79	7	37	3	50	6	14	4	55	8	18	2	41
16	4	96	3	14	6	57	1	47	8	65	5	75	9	79	2	71	7	60	10	22
17	2	31	8	47	9	58	4	32	5	44	6	58	7	34	1	33	3	69	10	51
18	2	44	8	40	3	17	1	62	9	66	7	15	4	29	10	38	6	8	5	97
19	3	58	4	50	5	63	10	87	1	57	7	21	8	57	9	32	2	39	6	20
20	2	85	1	84	6	56	4	61	10	15	8	70	9	30	3	90	7	67	5	20

Tabela B.10 Instância LA31

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	5	21	8	26	10	16	3	34	4	55	9	52	6	95	7	71	2	21	1	53
2	9	77	6	98	2	42	8	66	3	31	4	39	7	77	10	79	5	55	1	12
3	3	64	5	92	4	34	2	19	9	62	7	54	8	43	1	83	10	79	6	37
4	1	93	9	24	4	69	8	38	6	77	3	87	5	60	7	41	2	87	10	83
5	10	77	1	44	5	96	9	79	7	75	3	98	6	25	4	17	8	43	2	49
6	4	76	3	35	6	28	1	95	8	95	5	61	9	35	2	7	7	9	10	10
7	2	91	8	27	9	50	4	16	5	28	6	59	7	52	1	46	3	59	10	43
8	2	45	8	71	3	39	1	87	9	14	7	54	4	41	10	43	6	9	5	20
9	3	37	4	26	5	33	10	42	1	78	7	89	8	8	9	66	2	28	6	33
10	2	74	1	69	6	84	4	27	10	81	8	45	9	69	3	94	7	78	5	96
11	6	76	8	32	7	18	1	20	4	87	3	17	10	25	5	24	2	31	9	81
12	10	97	9	90	6	28	8	86	1	58	2	72	3	23	7	76	4	99	5	45
13	10	48	6	27	7	67	8	62	5	98	1	42	2	46	9	27	4	48	3	17
14	10	80	4	19	6	28	2	12	5	94	7	63	8	98	9	50	1	80	3	50
15	3	50	2	41	5	61	9	79	6	14	10	72	8	18	4	55	7	37	1	75
16	10	22	6	57	5	75	3	14	8	65	4	96	2	71	1	47	9	79	7	60
17	4	32	3	69	5	44	2	31	10	51	1	33	7	34	6	58	8	47	9	58
18	9	66	8	40	3	17	1	62	10	38	6	8	7	15	4	29	2	44	5	97
19	4	50	3	58	7	21	5	63	8	57	9	32	6	20	10	87	1	57	2	39
20	5	20	7	67	2	85	3	90	8	70	1	84	9	30	6	56	4	61	10	15
21	7	29	1	82	5	18	4	38	8	21	9	50	2	23	6	84	3	45	10	41
22	4	54	10	37	7	62	6	16	1	52	9	57	5	54	3	38	8	74	2	52
23	5	79	2	61	9	11	1	81	8	89	7	89	6	57	4	68	10	81	3	30
24	10	24	2	66	5	32	4	33	9	8	3	20	7	84	1	91	8	55	6	20
25	4	54	3	64	7	83	10	40	8	8	1	7	5	19	6	56	2	39	9	7
26	2	6	5	74	1	63	3	64	10	15	7	42	8	98	9	61	6	40	4	91
27	2	80	4	75	1	26	3	87	10	22	8	39	9	24	5	75	7	44	6	6
28	6	8	4	79	7	61	2	15	1	12	8	43	9	26	10	22	3	20	5	80
29	2	36	1	63	8	10	5	22	4	96	6	40	10	5	9	18	7	33	3	62
30	5	8	9	15	3	64	4	95	2	96	7	38	8	18	10	23	6	64	1	89

Tabela B.11 Instância LA36

Jobs	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
1	5	21	4	55	7	71	15	98	11	12	3	34	10	16	2	21	1	53	8	26
2	12	54	5	83	2	77	8	64	9	34	15	79	13	43	1	55	4	77	7	19
3	10	83	6	77	3	87	8	38	5	60	13	98	1	93	14	17	7	41	11	44
4	6	77	1	96	10	28	7	7	5	95	14	35	8	35	9	76	12	9	13	95
5	11	87	5	28	9	50	3	59	1	46	12	45	15	9	10	43	7	52	8	27
6	1	20	3	71	5	78	14	66	4	14	13	8	15	42	7	28	2	54	10	33
7	9	69	5	96	13	17	1	69	8	45	12	31	7	78	11	20	4	27	14	87
8	5	58	14	90	12	76	4	81	8	23	10	28	2	18	3	32	13	86	9	99
9	6	27	2	46	7	67	9	27	14	19	11	80	3	17	4	48	8	62	12	12
10	12	37	6	80	5	75	9	55	8	50	1	94	10	14	7	41	15	72	4	50
11	8	65	4	96	1	47	5	75	13	69	15	58	11	33	2	71	10	22	14	32
12	2	34	3	47	4	58	6	51	5	62	7	44	10	8	8	17	11	97	9	29
13	4	50	8	57	14	61	6	20	12	85	13	90	3	58	5	63	11	84	2	39
14	10	84	8	45	6	15	15	41	11	18	5	82	12	29	3	70	2	67	4	30
15	10	37	11	81	12	61	15	57	9	57	1	52	8	74	7	62	13	30	2	52

Tabela B.11 Instância LA36 (cont.)

M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i	M_j	p_i
9	52	6	95	13	31	12	42	14	39
10	37	6	79	11	92	14	62	3	66
4	69	12	49	9	24	2	87	15	25
3	43	2	75	11	61	15	10	4	79
2	91	14	41	4	16	6	59	13	39
12	89	9	26	8	37	11	33	6	43
2	74	6	84	15	76	3	94	10	81
15	97	1	24	11	45	7	72	6	25
15	28	5	98	1	42	10	48	13	50
11	61	14	79	3	98	13	18	2	63
6	57	9	79	3	14	12	31	7	60
12	15	14	66	13	40	1	44	15	38
10	87	7	21	15	56	9	32	1	57
14	50	7	23	1	20	13	21	9	38
3	38	14	68	5	54	4	54	6	16