Solving the Uncapacitated Facility Location Problem using Fusion Moves: Applications in Computer Vision

Author:
Vasco Tavares Salavessa Gama Mota

Supervisors:
Prof. Dr. João P. Barreto
Prof. Dr. Gabriel Falcão

Master Thesis in Electrical and Computer Engineering

February 2015

Resumo

O problema da Localização de Instalações Não-Limitadas (LINL) é um problema versátil, com muitas aplicações, nomeadamente na área de Visão por Computador ([1], [2] e [3]). Delong et al.[4] apresentam um novo algoritmo para resolver o problema da LINL, o Fast Fusion Moves (FFM). Este utiliza aplicações da teoria de grafos para atingir aproximações sub-óptimas das soluções do problema da LINL. Aplicações da teoria de grafos são muito utilizadas em problemas de Visão por Computador ([5] and [6]).

Nesta tese, comparamos o algoritmo FFM com um algoritmo *state of the art*, da autoria de Lazic et al. [7], o Message Passing (MP). Esta comparação é feita em dois problemas de Visão por Computador: detecção de planos [2] e calibração de câmaras [3]. Mostramos que o FFM consegue ser 10 vezes mais rápido que o MP, mas que a qualidade das suas soluções não atinge o nível do MP. Como o FFM é altamente dependente do esquema de fusão, propomos novos esquemas de fusão, que aumentam a qualidade e rapidez do algoritmo. Propomos ainda um algoritmo híbrido que combina a velocidade do FFM com a qualidade do MP.

Palavras Chave

Localização de Instalações Não-Limitado, Fast Fusion Moves, Teoria de Grafos, Cobertura de Vértices Mínima, Fluxo Máximo de um grafo, Corte Mínimo de um grafo

Abstract

The Uncapactitated Facility Location (UFL) Problem is a versatile optimization problem, used in some works on Computer Vision ([1], [2] and [3]). Delong et al. [4] propose a new solver, the Fast Fusion Moves (FFM), using graphical methods to approach suboptimal solutions of the UFL Problem. Graphical techniques are nowadays a widespread tool to solve Computer Vision Problems ([5] and [6]).

On this work, we compare the FFM algorithm with a state of the art UFL solver from Lazic et al.[7], the Message Passing (MP) algorithm. This comparison is done using two Computer Vision Problems: plane detection [2] and camera calibration [3]. We show that the FFM algorithm is 10$\times$ faster than the MP, but its solutions do not show the same level of quality. As the FFM algorithm is highly dependent of the fusion scheme, we propose new fusion schemes that improve its quality and speed. We also propose an hybrid algorithm that combines the speed of the FFM algorithm with the quality of the MP algorithm.

Keywords

Ucapactitated Facility Location, Fast Fusion Moves, Graphical Methods, Optimization, Computer Vision, Vertex Cover, Min Cut, Max Flow

Acknowledgements

First, I would like to thank my supervisors, Professor João Barreto and Professor Gabriel Falcão. To Professor João Barreto I would to thank his patience, his attention to detail and his work ethic. To Professor Gabriel Falcão I thank his motivational skills, his curiosity and his availability. After working with both of them, I am not just sure that I became a better professional, I know I became a better person.

Before my advisers, there was a person that motivated me to work with graphical methods. To Professor João Gouveia I must thank his availability and helpfulness.

I would like to thank Pedro Rodrigues, Carolina Raposo, Frederico Vasconcelos, Michel Antunes and Rui Melo for all the interesting discussions we had at the lab. These were the first people I would turn to when in doubt and they never refused to spend some of their time to enlighten me.

To all my friends, inside and outside the major, I have to thank them for all the care and support. I cannot count the amount of times you had to hear me ranting about this thesis. In special, I would like to thank Rachel Hanne for the revisions of my thesis, that undoubtedly improved this text beyond my capabilities. I also want to thank André Pinto, who accepted the challenge of creating the cover for this work. I could not have asked for a better cover.

Finally, I have to thank my family. To my amazing sister, who always kept me down to earth and helped me in more ways than I will ever be able to thank her. And to my parents, who were always there for me. They backed my every step, and if today I am able to produce this work, this is mostly due to their efforts. I hope one day I might inspire someone as much as they inspire me.

Contents

List of Figures

List of Acronyms

CV Computer Vision

FL Facility Location

FFM Fast Fusion Moves

GT Groundtruth

MAP maximum-a-posteriori

minWVC Minimum Weighted Vertex Cover

MP Message Passing

MRF Markov Random Fields

PB Pseudo Boolean

PPR Piecewise-Planar Reconstruction

SPLP Simple Plant Location Problem

UFL Uncapacitated Facility Location

UIC Unsupervised Intrinsic Calibration

VC Vertex Cover

Chapter 1

Introduction

1.1 Motivation and Objectives

Graphical methods are a useful tool for Computer Vision (CV) problems. Sinha [5] and Kolmogorov [8] transformed several CV problems into graph problems and performed optimization over them. This transformation usually involves formulating the CV instances as energy minimization problems. A corresponding graph is created and then a minimization routine is performed over it. As an example, an energy function that can be transformed in a minimum cut graph problem can be solved in polynomial time [5].

The Uncapacitated Facility Location (UFL) problem is a classical problem in the Operational Research field. It can be used to label data, to determine the best set of facilities to serve a group of customers, or as a clustering tool. As facilities and customers can be abstract, the applications of the UFL are endless. It is a problem that allows many different strategies to solve it, one of them being graphical methods. The UFL has very recently found its place in CV problems, like space segmentation [1] or the works of our group ([2] and [3]) .

The focus of this work is a graphical method to solve the UFL: the Fast Fusion Moves (FFM) method, proposed by Delong et al. [4]. We compare this solver to the Message Passing (MP) algorithm created by Lazic et al. [7]. The MP algorithm also uses graphical methods but it requires greater computational effort.

The objective of this work is evaluating if the FFM algorithm is a viable alternative to the MP, validating it in two practical CV problems: plane detection [2] and camera calibration [3].

1.2 The Uncapacitated Facility Location Problem

The UFL is one of the most widely studied discrete location problems. The *Facility Location* term in Uncapacitated Facility Location (UFL) stands for the problem of determining the best location for a given facility (or the best locations for a given set of facilities) given some constraints about the environment where it can be placed.

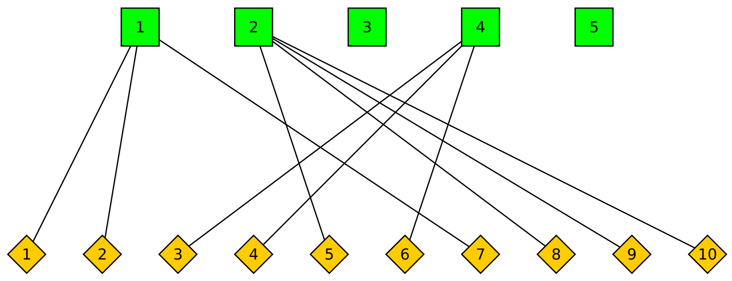


Figure 1.1: Example of an assignment between facilities (green squares) and customers (yellow diamonds).

The UFL problem can be stated as follows: given a set of customers, a set of potential facilities that can be opened to serve customers, the cost of opening each facility, and the distances between customers and facilities, open a subset of facilities and assign customers to one facility each, such that the sum of facility costs and customer-facility distances is minimized. Figure 1.1 shows an example of an assignment between facilities and customers. Facilities can also be named *labels*.

Consider a set of of customers $\mathcal{C}$ and a set of facilities $\mathcal{F}$. Consider also a cost $c_{ij} : \mathcal{C} \times \mathcal{F} \to \mathbb{R}$ of assigning customer $i \in \mathcal{C}$ to facility $j \in \mathcal{F}$. Each customer is assigned to exactly one facility. Also, consider the opening cost $f_j : \mathcal{F} \to \mathbb{R}$ of a facility $j \in \mathcal{F}$. Consider too the set $\mathcal{C}_j \subset \mathcal{C}$ of customers assigned to facility $j \in \mathcal{F}$ ($\mathcal{C}_j = \{i \in \mathcal{C} : i \text{ is assigned to facility } j \in \mathcal{F}\}$). Finally, there is the set of opened facilities, $\mathcal{F}^O \subset \mathcal{F}$, ($\mathcal{F}^O = \{j \in \mathcal{F} : |\mathcal{C}_j| > 0\}$).

Our problem is following minimization:

$$\min \sum_{j \in \mathcal{F}^O} \sum_{i \in \mathcal{C}_j} c_{ij} + \sum_{j \in \mathcal{F}^O} f_j \tag{1.1}$$

The *Uncapacitated* term is used in opposition to the Capacitated Facility Location Problem, where the facilities have a limit on the amount of customers they can serve. In the Uncapacitated version, such a constraint does not exist.

1.3 Applications of UFL in Computer Vision

If we think about facilities as models and the customers as a set of data, the UFL framework can be used to solve multi-model fitting. Multi-model fitting is a re-current problem in CV applications. It is a chicken and egg problem from which data is used to create models which are then used to cluster data. It has been traditionally solved using a greedy approach such as sequential RANSAC. Recent works [9] have shown that non-local approaches are advantageous which brings in graphical methods in general, and UFL in particular, as plausible solutions.

Other possible application of the UFL is clustering data. The *k-means*, which is a basic clustering technique, can be modeled as an instance of the UFL, where the set of customers and facilities is the same and exactly k facilities are opened.

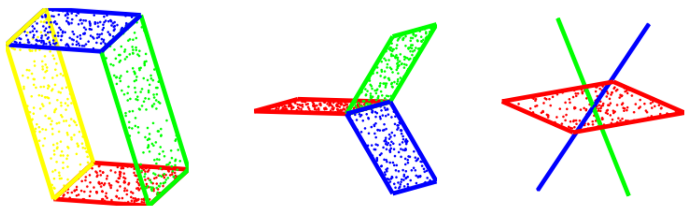


Figure 1.2: Examples of data lying on multiple linear subspaces (source: [1]).

Lazic et al. [1] is one of the few examples of casting a CV problem as a UFL problem. In this work, Lazic proposes a new method for space segmentation using UFL. Space Segmentation is a technique used on several CV applications as motion segmentation in video or structure-from-motion problems. In this UFL model, the clients are data points and the facilities are linear subspaces, used to describe the data points. Figure 1.2 shows an example of data points (clients) being fit by linear subspaces (facilities).

The objective of this space segmentation algorithm is to describe the high-dimensional input data with low-dimensional linear subspaces. The more complex the linear subspaces are, the better they will describe the data (everyone can fit three points in a parabola, but will they fit on a line?). So, the facility cost associated to the candidate linear subspaces will grow with their complexity.

Other examples of applications of UFL in CV are the works of our group: Melo [3], and Raposo [2].

1.4 Contributions

This work provides the following contributions:

- We implemented the FFM algorithm originally described in [4] and applied it to two concrete CV problems.

- To our knowledge, this is the first comparative analysis of the FFM algorithm against the MP algorithm, concluding that the FFM algorithm is substantially faster but does not always converge to the correct solution.

- We proposed new versions of FFM that enhance the quality and speed of the algorithm, including new fusion schemes and an hybrid version that uses FFM and MP

- We concluded that the quality of the FFM's solutions is dependent on the fusion scheme used.

1.5 Dissertation Outline

This work is structured in seven chapters. Chapter 2 gives a background on graph theory. Chapter 3 presents the formulation of the UFL problem, a brief historical review and an overview of the MP algorithm. Chapter 4 describes the FFM algorithm. Chapter 5 presents two applications (Piecewise Planar Reconstruction and the Unsupervised Intrinsic Calibration) and the results of using the FFM algorithm to solve them. Chapter 6 provides techniques to further enhance the quality and speed of FFM. Finally, chapter 7 lists our conclusions about the FFM algorithm and suggestions for Future Work.

Chapter 2

Background on Graphical Methods

2.1 Basic Concepts

The next few topics review the concepts of graph, bipartite graph, graph cut and graph flow.

2.1.1 Graphs

A graph $G(\mathcal{V}, \mathcal{E})$ is a pair of sets: the set of vertices $\mathcal{V}$ and the set of edges $\mathcal{E}$ that connects some of those vertices. $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$.

An example of a graph is given in Figure 2.1. In this graph $\mathcal{V} = \{1, 2, 3, 4\}$, $\mathcal{E} = \{\{1, 2\}, \{1, 4\}, \{2, 3\}, \{2, 4\}\}$.

2.1.2 Bipartite Graphs

A graph $G(\mathcal{V}, \mathcal{E})$ is bipartite if and only if $\mathcal{V}$ can be separated in two subsets $\mathcal{V}_0$ and $\mathcal{V}_1$ in such way that all the edges in the graph go from $\mathcal{V}_0$ to $\mathcal{V}_1$.

Formally,

$$G(\mathcal{V}, \mathcal{E}) \text{ is bipartite iff } \exists \mathcal{V}_0, \mathcal{V}_1 \subset \mathcal{V}, \mathcal{V}_0 \cup \mathcal{V}_1 = \mathcal{V}, \mathcal{V}_0 \cap \mathcal{V}_1 = \{\} \text{ s.t. } \forall \{u, v\} \in \mathcal{E}, u \in \mathcal{V}_0, v \in \mathcal{V}_1 \tag{2.1}$$

This means that there are no two vertices in the same subset $\mathcal{V}_0$ or $\mathcal{V}_1$ connected by an edge.

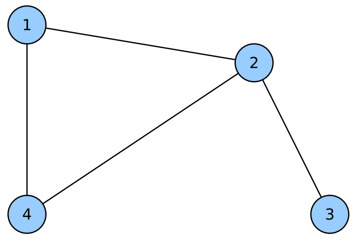


Figure 2.1: Example of a graph.

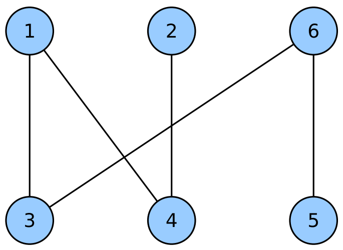


Figure 2.2: Example of a bipartite graph. $\mathcal{V}_0 = \{1, 2, 6\}$ and $\mathcal{V}_1 = \{3, 4, 5\}$.

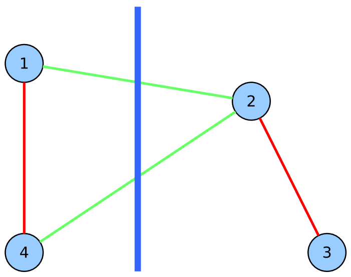


Figure 2.3: Example of a cut of the previously shown graph. Green edges belong to the cut, the red ones do not.

2.1.3 Graph Cuts

A cut $C = (S, T)$ is a partition of the vertices of a graph into two disjoint subsets $S, T \subset \mathcal{V}$, $S \cup T = \mathcal{V}$, $S \cap T = \emptyset$. The cut-set of a cut $C = (S, T)$ is the set of edges $\mathcal{E}_C = \{\{u, v\} \in \mathcal{E} | u \in S, v \in T\}$. The number of edges in the cut-set, $|\mathcal{E}_C|$ can also be called the *cut size*.

An example of a graph is given in Figure 2.3. In this example, $S = \{1, 4\}$ and $T = \{2, 3\}$. $\mathcal{E}_C = \{\{1, 2\}, \{2, 4\}\}$.

If the edges have capacities, $c_{\{u,v\}} \in \mathbb{R}_0^+$ associated with them, the cut size is the sum of the capacities of the edges in $\mathcal{E}_C$ ($c = \sum_{\{u,v\} \in \mathcal{E}_C} c_{\{u,v\}}$).

2.1.4 Graph Flow

The edges of a graph can either be directed or undirected. If they are directed, an edge goes *from* one vertex *to* another vertex. Also, edges became ordered pairs: edges $e = (u, v) \in \mathcal{E}$ and $e' = (v, u) \in \mathcal{E}$ are different (because e goes from u to v and e' goes the opposite way). In undirected graphs, there is no difference between e and e'.

Direct graphs can represent *flow networks* (as in Figure 2.4), as long as some requirements are fulfilled:

- Each edge $e = (u, v) \in \mathcal{E}$ has a capacity $c(u, v) \in \mathbb{R}_0^+$
- If no edge (u, v) exists, than $c(u, v) = 0$

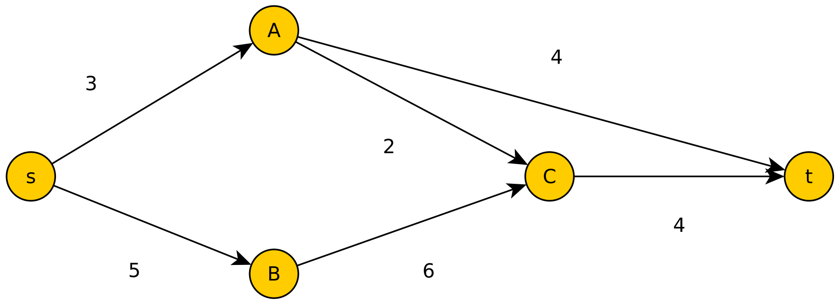


Figure 2.4: Example of a flow network. Each edge has its capacity next to it.

- There are two special vertices: the source s and the sink t. No edges go into the source ($\nexists v \in \mathcal{V}$ s.t. $(v, s) \in \mathcal{E}$) and no edges come out of the sink ($\nexists v \in \mathcal{V}$ s.t. $(t, v) \in \mathcal{E}$).

A *flow* is a function $f : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that verifies two properties:

1. Capacity Constraint:
$$\forall u, v \in \mathcal{V}, 0 \leq f(u, v) \leq c(u, v) \tag{2.2}$$

2. Flow Conservation Constraint:
$$\forall u \in \mathcal{V} \backslash \{s, t\}, \sum_{v \in \mathcal{V}} f(v, u) = \sum_{v \in \mathcal{V}} f(u, v) \tag{2.3}$$

The Capacity Constraint ensures that the flow over any edge never exceeds its capacity. The Flow Conservation Constraint ensure that all the flow that goes into a vertex ($\sum_{v \in \mathcal{V}} f(v, u)$) is equal to the amount that leaves it ($\sum_{v \in \mathcal{V}} f(u, v)$), except for the source (s) and the sink (t) nodes.

The value of the flow over a network, $|f|$, is defined as the total flow that leaves the source node.

$$|f| = \sum_{v \in \mathcal{V}} f(s, v) \tag{2.4}$$

Graph cuts in flow networks

A cut over a flow network must separate the sink from the source, i.e., if $c(S, T)$ is a cut over a network flow $G(\mathcal{V}, \mathcal{E})$, then s (the source node) must be in S and t (the sink node) must be in T. The size of a cut in a flow network is given by the capacity of the edges that go from vertices in S to vertices in T, as formalized by Equation 2.5.

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v) \tag{2.5}$$

The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G [10].

2.2 Minimum Vertex Cover Problem

A Vertex Cover (VC) ($\mathcal{V}_C$) is a subset of $\mathcal{V}$, on which each edge of the graph is incident to at least one vertex of the set. Formally,

$$\mathcal{V}_C \subset \mathcal{V} \text{ is a cover iff } \forall e = \{u, v\} \in \mathcal{E}, u \in \mathcal{V}_C \vee v \in \mathcal{V}_C \tag{2.6}$$

This means that there is not a single edge $e = \{u, v\} \in \mathcal{E}$ that verifies that $u \notin \mathcal{V}_C \wedge v \notin \mathcal{V}_C$.

The Minimum Vertex Cover Problem consists in minimizing $|\mathcal{V}_C|$. We want to determine the smallest subset of $\mathcal{V}$ that still is a vertex cover. It is an NP-complete problem. However, if the graph is bipartite, the problem is solvable in polynomial time.

2.2.1 Minimum Weighted Vertex Cover Problem

On the Minimum Weighted Vertex Cover (minWVC) problem, each vertex $v \in \mathcal{V}$ has a weight $w_v \in \mathbb{R}_0^+$ associated with it. The problem consists on minimizing not the number of vertices on the cover as previously, but to minimize the sum of their weights.

The problem can be formalized as the minimization of a pseudo-boolean function. We associate a boolean variable $x_v \in \mathbb{B}(= \{0, 1\})$ to each vertex $v \in \mathcal{V}$. $x_v = 1$ if $v \in \mathcal{V}_C$ and $x_v = 0$ otherwise. The minimization that formalizes the minWVC is the following:

$$\min \sum_{v \in V} w_v x_v \tag{2.7}$$

$$\text{s.t. } \forall \{u, v\} \in \mathcal{E}, x_u + x_v \geq 1 \tag{2.8}$$

$$x_v \in \mathbb{B}, \forall v \in V \tag{2.9}$$

The condition in Equation 2.8 formalizes that all edges of the graph must be incident to at least one vertex in $\mathcal{V}_C$. Since $x_u = 1$ if $u \in \mathcal{V}_C$ and $x_u = 0$ otherwise, to verify Equation 2.8 either u or v must be in $\mathcal{V}_C$.

2.3 Max Flow / Min Cut Problem

One of the reasons that makes graphical methods so popular in CV is the ability to transform minimization problems into graph cut problems. The idea is to build a graph from the energy function we want to minimize in such a way that the size of the minimum cut of that graph is the same as the energy function's minimum (or, at least, there is a known relation between the two).

Kolmogorov et al. [8] characterized pseudo-boolean energy functions and determined how to build the corresponding graphs. As many problems in CV can be casted as the

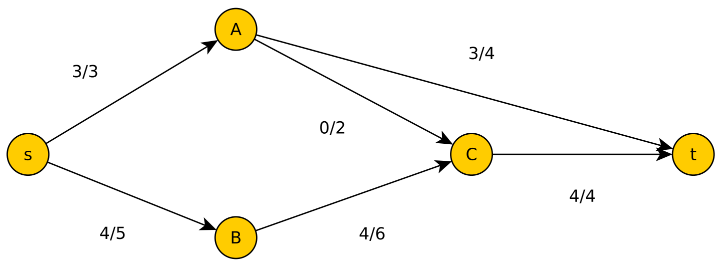


Figure 2.5: The maximum flow over the flow network previously presented.

minimization of an energy function, graphical solutions prove to be a popular tool.

The max-flow min-cut theorem [10] tells us that if f is a maximum flow in a flow network G (meaning that $|f|$ is biggest amongst all possible flows in G), then $|f| = c(S,T)$ for some cut (S,T) of G. Since the flow is a lower bound of the cut, $c(S,T)$ is a minimum cut of G. So, we can transform min-cut problems into max-flow problems (Section 2.1.4) in order to solve them.

Consider Figure 2.5, that shows the maximum flow over the flow network previously shown in Figure 2.4. Each edge has its flow and capacity presented as *flow/capacity*. In this example, the flow is 7, since $|f| = \sum_{v \in \mathcal{V}} f(s,v) = f(s,A) + f(s,B) = 3 + 4 = 7$ (Eq. 2.4).

Determining a graph's cut is a different problem from finding which edges lie on that cut. Similarly, determining a function's minimum is a different problem from finding the location of the minimum. However, after the maximum flow of a graph is calculated, finding its minimum cuts is straightforward. To determine partitions (S,T) that have minimum cuts, we do a breadth first search starting from the source node until we reach saturated edges (those whose flow is equal to their capacity). The set of saturated edges we find is a minimum cut.

Consider the example in Figure 2.5. If we do a breadth first search for saturated edges, we find the edges (s,A) and (C,t). This tells us the cut is $S = \{s,B,C\}$ and $T = \{A,t\}$. The capacity of this cut is (Eq. 2.5) $c(S,T) = c(s,A) + c(s,t) + c(B,A) + c(B,t) + c(C,A) + c(C,t) = 3 + 0 + 0 + 0 + 0 + 4 = 7$, which is equal to the flow previously calculated. Notice how the capacity of the edege (A,C) does not enter this calculation, since it goes from a vertex in T to a vertex in S.

2.4 Graphical Methods for Computer Vision

Under some applications, graph's vertices might represent the arguments of a function. An example of a such use is factor graphs. For more insight on factor graphs read the tutorial from Kschischang et al. [11].

In other applications, vertices represent pixels of an image and the edges connecting neighbor pixels have weights associated to them. For example, in Image Segmentation, each pixel is associated to a vertex and the weights of the edges between adjacent pixels represent their similarity.

The work of Sinha et al. [5], in which he proposes a framework for Energy Minimization using graph cuts, is quite relevant for this work. His framework models Markov Random Fields (MRF) as graphs, and solves them using max-flow algorithms. His work's relevance is measurable by the amount of problems modeled as MRF: texture synthesis, stereo, or image restoration are just some examples of classic CV problems that Sinha, through his framework, enabled to be solved using graphical methods. Besides Computer Vision, Sinha presents applications of graphical methods to Machine Learning and Computer Graphics.

Kolmogorov presented severall works on the subject as well. With Boykov [6], they gathered a series of CV problems and then solved them using graph cuts. In the same paper, they present an algorithm to solve min graph cut problems (see Section 2.1.3) that we used on our work. According to them, it was Greig et al. [12] the first to use min-cut/max-flow algorithms to minimize certain important energy functions in CV. With Zabih[8], Kolmogorov goes on to talk about generic energy functions and how those can be translated into graphs, in order to apply min-cut/max-flow solvers to compute the original functions minimal energy. Boykov had already presented graphical solutions for methods used in labeling problems like expansion and swap moves (for the labels). In his work [13] with Veksler and Zabih, they present new algorithms for alpha-expansion and alpha-beta-swap techniques that use graph cuts.

The work of Boykov and Kolmogorov [6], shows some examples of problems that can be solved using graphical methods:

- Image Segmentation
- Image Restoration
- Stereo
- Shape Reconstruction
- Object Recognition
- Texture Synthesis

Chapter 3

The UFL Problem

3.1 History of the Facility Location Problem

The Facility Location Problem has its origins in Economics and Operational Research. In this problem, one tries to determine the best place for an industrial facility or plant. That is why this problem is also called Simple Plant Location Problem (SPLP). It was Spielberg [14] who, in 1969, first used the term SPLP. The placement of an industrial facility is constrained by several economical factors and also the cost of transporting the commodities to customers. Krarup and Pruzan [15] did a very complete survey about the origins of this problem.

The first work attempting to solve location problems was by Evangelista Torricelli, an Italian physicist and mathematician. In 1648, he worked on the minimization of a set of distances to a set of points, in a Cartesian perspective. Other famous mathematicians have worked on the subject, like Pierre de Fermat.

But it was Alfred Weber, a German economist, that made the problem quite famous. In 1909, he defined the three main factors that influence the location of an industrial facility: *material index* (the ratio of weights of the raw materials to the finished product), *labor distortion* (sources of lower cost labor may justify greater transport distances) and *agglomeration* (the number of firms in the area). His work was so relevant that he is now considered the father of Industrial Location Theory. Also, the Facility Location Problem version where there is only one facility to be located is called the *Webber Problem* nowadays.

The first work to formulate the SPLP in its modern notation was from Balinsky and Wolfe [16] . Unfortunately, this paper was lost [15] but it was cited on another paper by Balinsky [17] three years later. One of the first solvers for UFL was a work by Kuehn and Hamburger [18] and it was what we now call a "greedy algorithm". The algorithm would add one facility at a time, to the set of opened facilities, opening the facility that gave the best improvement to the cost function. The algorithm would stop when the opening of any closed facility would not improve the cost function.

Reiter and Sherman [19] proposed a different idea. Imagine a $|\mathcal{F}|$-dimensional unitary hypercube. Its vertices' coordinates would be $|\mathcal{F}|$-dimensional vectors of 1s and 0s. Each element in the vertex corresponds to a facility, being 1 if such facility is opened, and 0 otherwise. What the algorithm would do is evaluate how the cost function would be for the m neighbour vertices and move to the most favorable one. Each of this moves corresponds to either open a closed facility or close an opened one.

Given its economical background, there are more versions of the Facility (or Plant) Location Problem that take in account other factors. For instance, the cost of each facility might be influenced by the customer's need for a certain commodity. Some formulations take in account the "economy-of-scale" or other factors. This leads to more complex problems, most of which can not be solved with the tools we present in this work.

UFL is known as the *Simple* Plant Location Problem precisely because it is the simplest formulation that takes in account less economical factors. However, the term *Simple* is misleading, as Guignard and Spielberg wrote:

> The SPLP is one of the simplest mixed integer problems which exhibits all the typical combinatorial difficulties of mixed (0-1) programming and at the same time has a structure that invites the application of various specialized techniques [20].

3.2 Solving the UFL Problem

Except for trivial cases (like the Weber problem [21], where there is only one facility to be positioned), the UFL problem is NP-Hard [15]. In the most general case, it cannot even be approximated to within any constant factor.

Our approach, as proposed by Andrew Delong, Olga Veksler and Yuri Boykov in their work [4] implements a strategy where several bad solutions are combined (*fused*) into a better solution. Each of these fusion moves is casted as a minWVC problem. The **minVC!** (**minVC!**) problem is NP-hard [22]. However, there exist 2-approximation algorithms for generic weighted vertex cover problems. If the graph on the minWVC problem is bipartite, then max-flow algorithms can be used to solve the problem, which run in polynomial time.

3.3 Pseudo Boolean Formulation of UFL

Equation 1.1 might be intuitive but it is not computationally useful. We now present the Pseudo Boolean (PB) formulation of UFL.

A function f_B is a *Boolean Function* if and only if f_B is defined as $f_B : \mathbb{B}^n \to \mathbb{B}, n \in \mathbb{N}$ ($\mathbb{B} = \{0, 1\}$). f_{PB} is a *PB Function* if its codomain is $\mathbb{R}$ instead of $\mathbb{B}$ (i.e., $f_{PB} : \mathbb{B}^n \to \mathbb{R}, n \in \mathbb{N}$).

This PB version of the UFL problem takes two sets of boolean variables as input:

- variables x_{ij} represent the assignment between customers and facilities: $x_{ij} = 1$ if client i is assigned to facility j and $x_{ij} = 0$ otherwise. The vector of x_{ij} variables is $\mathbf{x}$.

- variables y_j represent the state of facilities : $y_j = 1$ if facility j is opened (i.e., if there is at least one costumer assigned to that facility) and $y_j = 0$ otherwise.

The PB problem statement is the following:

$$\min_{\mathbf{x}} \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{F}} c_{ij} x_{ij} + \sum_{j \in \mathcal{F}} f_j y_j \tag{3.1}$$

constrained by the following conditions

$$\sum_{j \in \mathcal{F}} x_{ij} = 1, \forall i \in \mathcal{C} \tag{3.2}$$

$$y_j \geq x_{ij}, \forall i \in \mathcal{C}, j \in \mathcal{F} \tag{3.3}$$

$$x_{ij}, y_j \in \mathbb{B}, \forall i \in \mathcal{C}, j \in \mathcal{F} \tag{3.4}$$

Condition 3.2 ensures that all clients are assigned to a facility. Condition 3.3 ensures that, if there is at least one client assigned to a facility, then that facility must be opened (forcing $y_j = 1$ if there is at least one $x_{ij} = 1$). Finally, condition 3.4 enforces x_{ij} and y_j variables to be boolean variables ($x \in \mathbb{B} \Leftrightarrow x \in \{0, 1\}$).

Problems that follow the PB formulation can be solved using integer programming techniques, so this is also called the *integer formulation*.

3.4 Message Passing Algorithms to Solve UFL

Lazic et al.[7] propose a probabilistic inference on a graphical model of the UFL. They formulate the UFL as a maximum-a-posteriori (MAP) problem, where x_{ij} and y_j are treated as hidden variables. The joint log-likelihood of the hidden variables corresponds to the UFL problem objective.

The MAP problem is solved using a MP algorithm over a factor graph. Read [11] for more insight on factor graphs, the sum-product algorithm and the log-domain transformation.

Usually, factor graphs are used to represent factorisable functions. However, in this case, we work on the log-domain, so each factor is a parcel of a sum. The factor graph representation of the UFL is in Figure 3.1. The MAP UFL problem becomes

$$max_{\mathbf{x}} \sum_{ij} \theta_{ij}(x_{ij}) + \sum_{j} \theta_j^F(\mathbf{x_{:j}}) + \sum_{i} \theta_i^C(\mathbf{x_{i:}}) \tag{3.5}$$

where the sub-functions are the following

$$\theta_{ij}(x_{ij}) = -c_{ij} x_{ij} \tag{3.6}$$

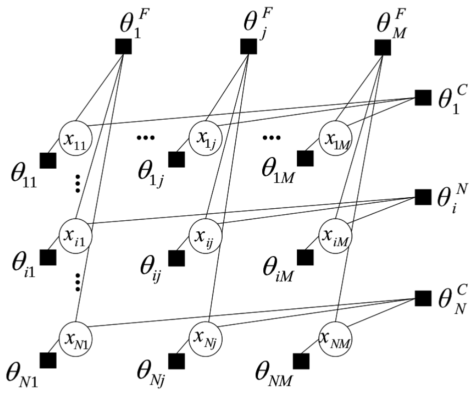


Figure 3.1: Factor graph representation of the UFL problem. Source: [7].

$$\theta_j^F(\mathbf{x_{:j}}) = \begin{cases} -f_j, & \sum_i x_{ij} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

$$\theta_j^C(\mathbf{x_{i:}}) = \begin{cases} 0, & \sum_j x_{ij} = 1 \\ -\infty, & \text{otherwise} \end{cases} \tag{3.8}$$

Vectors $\mathbf{x_{:j}}$ and $\mathbf{x_{i:}}$ are, respectively, the vector of decision variables involving facility j and vector of decision variables involving client i. So, $|\mathbf{x_{:j}}| = |\mathcal{C}|$ and $|\mathbf{x_{i:}}| = |\mathcal{F}|$. Functions $\theta_j^C(\mathbf{x_{i:}})$ (one for each facility) ensure that exactly one facility is assigned to each customer (like Constraint 3.2). Functions $\theta_j^F(\mathbf{x_{:j}})$ incorporate the facility costs and ensures that Condition 3.3 is valid.

So, we are able to transform a problem with one function with $|\mathcal{C}| \times |\mathcal{F}|$ variables to a problem of $|\mathcal{F}| + |\mathcal{C}|$ functions with $|\mathcal{C}|$ or $|\mathcal{F}|$ arguments each.

There are also the Functions $\theta_{ij}(x_{ij})$ (one for each pair of customers and facilities), but this functions do not need to be updated, only in the end, so they don't contribute additional computational weight.

The function's values are negative (or zero) because we are maximizing the symmetric of Eq. 3.1.

3.4.1 Message Passing Algorithm

With the sub-functions set and the factor graph defined, we now need to solve the smaller optimization problems and propagate their results. In order to do that, we use the sum-product algorithm or, if we are in the log domain, the max-sum algorithm.

Max-sum iteratively updates the values between adjacent nodes of the factor graph. We consider that there are two types of messages: messages $m_{\theta\to x}(x)$ from functions to variables and messages $m_{x\to\theta}(x)$ from variables to functions. The first update the values of the variables involved in a maximization/minimization. The second inform functions that their arguments' value has changed.

This creates an iterative mechanism: functions are idle until they receive a message $m_{x\to\theta}(x)$ informing them that an argument's value was changed. Then, they perform optimization over the other arguments and write their resulting value trough a $m_{\theta \to x'}(x')$ message. At this point, a $m_{x' \to \theta'}(x')$ message is issued to functions adjacent to the updated variables.

The factor graph takes an important role on keeping track of what should be updated about changes of variables.

Chapter 4

Fast Fusion Moves for UFL

4.1 Introduction

Delong, Veksler and Boykov [4] present an alternative solver for the UFL. Their idea is to use fusion moves to solve the problem. Fusion moves are a process of creating a solution through the combination of two previous admissible solutions - by essentially fusing them. Their method is built in such way that the resulting solution is the best combination of the two inputs.

Unlike the MP algorithm, where we need to build a complex graph including all facilities and all clients, on this method we can combine several poor solutions that take in only a part of the facilities and stitch them together in order to create a better solution. In some ways, this algorithm resembles genetic algorithms in the sense that we combine poorer solutions in order to achieve a better one.

The algorithm will be presented in the following sections:

- Section 4.2 presents how the UFL problem is seen from the fusion perspective.
- Section 4.3 shows how the fusion moves can be transformed into a vertex cover sub-problem.
- Section 4.4 analyses the vertex cover sub-problem and shows how it can be approximated through a min-cut/max-flow problem.
- Section 4.5 presents the techniques used to solve the min-cut/max-flow problem.
- Section 4.6 gives an overall look of the problem, by showing how the solution from the min-cut/max-flow problem gives us the solution to the vertex cover problem, which in turn allows us to solve the initial fusion problem.

4.2 Fusion Formulation

In the PB formulation of the UFL problem (Eq. 3.1), the minimization was done over the $\mathbf{x}$ boolean vector, which has $|\mathcal{C}| \times |\mathcal{F}|$ elements.

Under the fusion solver, it is better to think of the decision variable as the vector $\mathbf{l}$, with $|\mathcal{C}|$ elements, each assigned to an element of $\mathcal{F}$, instead of the vector $\mathbf{x}$, with $|\mathcal{C}| \times |\mathcal{F}|$ elements, most of which are zeros. The relation between $\mathbf{l}$ and $\mathbf{x}$ is $l_i = j \Leftrightarrow x_{ij} = 1$. Since Condition 3.2 ensures that, for each customer i, there is only one facility j that verifies $x_{ij} = 1$, this is a direct transformation. The vector $\mathbf{l}$ can also be called the *labelling* of the customers.

Consider now the set $S \subseteq \mathcal{F}$ as the set of opened facilities in $\mathbf{l}$, c_{ij} as the distance between client i and facility j (or the cost of assigning client i to facility j) and f_j as the opening cost of facility j . The energy function becomes:

$$G(S) = \sum_{i \in \mathcal{C}} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \tag{4.1}$$

Consider an example of 6 costumers, $\mathcal{C} = \{1, 2, 3, 4, 5, 6\}$ and 5 facilities, $\mathcal{F} = \{A, B, C, D, E\}$. A possible labeling would be $\mathbf{l} = (B, B, C, D, B, A)$. For this labeling, the set S is $\{A, B, C, D\}$.

This labelling of customers has a central role in the Fusion solver because, at each step, two labellings ($\mathbf{l}^0$ and $\mathbf{l}^1$) are fused together, creating a new labelling ($\mathbf{l}'$) that receives the best aspects of the two labellings. As so, the energy of the resulting labelling $\mathbf{l}'$ is always at most the energy of the labellings being fused. Therefore, we can say that FFM is a monotonous method. The monotony of the method enables us to massively fuse labellings, with the assurance that the resulting labelling is never worst than the fused labellings.

But how are the labellings actually fused? And what is the relation between the resulting labelling and the initial ones? That's where the *fusion constraint* comes in place. The fusion constraint requires that, for a given customer, the label assigned to it in $\mathbf{l}'$ is either the label it had on $\mathbf{l}^0$ or in $\mathbf{l}^1$. Formally, $l'_i = l^0_i \vee l'_i = l^1_i, \forall i \in \mathcal{C}$.

S^0 and S^1 are the sets of labels present in the input labellings $\mathbf{l}^0$ and $\mathbf{l}^1$, respectively, and S' is the set of labels present in the resulting labeling $\mathbf{l}'$. The fusion operation can be thought as determining the set $S' \subseteq \{S^0 \cup S^1\}$ that verifies the fusion constraint.

Consider now the set $\mathbf{l}(S)$ of all the possible labellings given a set of labels S. $\mathbf{l}^0 \in \mathbf{l}(S^0)$ and $\mathbf{l}^1 \in \mathbf{l}(S^1)$. But what about the resulting labelling $\mathbf{l}'$? If S' verifies the fusion constraint, there must exist a labelling $\mathbf{l}' \in \mathbf{l}(S')$ that verifies that $l'_i = l^0_i \vee l'_i = l^1_i, \forall i \in \mathcal{C}$. The set $\mathcal{S}(\mathbf{l}^0, \mathbf{l}^1)$ contains all sets of facilities from which is possible to build labellings that verify the fusion constraint. So, $S' \in \mathcal{S}(\mathbf{l}^0, \mathbf{l}^1)$, where $\mathcal{S}(\mathbf{l}^0, \mathbf{l}^1)$ is given by Eq. 4.2:

$$\mathcal{S}(\mathbf{l}^0, \mathbf{l}^1) = \{S \subseteq \{S^0 \cup S^1\} | \exists \mathbf{l} \in \mathbf{l}(S) \ s.t. \ l_i \in \{l^0_i, l^1_i\}, \forall i \in \mathcal{C}\} \tag{4.2}$$

Given two initial labellings ($\mathbf{l}^0$ and $\mathbf{l}^1$), the fusion problem consists in determining the set $S \in \mathcal{S}(\mathbf{l}^0, \mathbf{l}^1)$ that minimizes the energy function below:

$$G(S) = \sum_{i \in \mathcal{C}} \min_{j \in \{l^0_i, l^1_i\} \cap S} c_{ij} + \sum_{j \in S} f_j, S \in \mathcal{S}(\mathbf{l}^0, \mathbf{l}^1) \tag{4.3}$$

Notice how the fusion constraint is integrated in this minimization through the condition

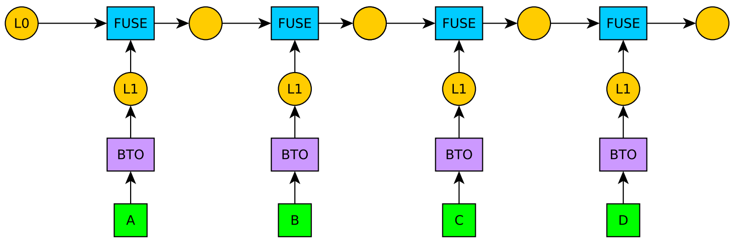


Figure 4.1: Linear fusion scheme. The green squares are the facilities, yellow circles are labellings, the purple rectangles represent the *betterThanOutlier* function (BTO) and blue rectangles are the *FUSE* function.

that $j \in \{l_i^0, l_i^1\} \cap S$.

4.2.1 Getting UFL Solution Using Fusion Moves

In order to solve the UFL using fusion moves, Delong et al. propose an iterative form of joining a new facility at each step. Figure 4.1 shows the scheme used. Labelling $\mathbf{l}^0$ is initialized with outlier labels $\emptyset$. At each step, labelling $\mathbf{l}^1$ is built using a new facility j_{new} through the *betterThanOutlier* function, the two labellings are fused and $\mathbf{l}^0$ is updated with the result.

```
Data: $\mathcal{C}, \mathcal{F}$
Result: Final labelling $\mathbf{l}$
for $i \in \mathcal{C}$ do
    $l_i^0 \leftarrow \emptyset$;
end
while stop criteria is not met do
    $j_{new} \leftarrow$ randomSampling($\mathcal{F}$);
    $\mathbf{l}^1 \leftarrow$ betterThanOutlier($j_{new}$, $\mathcal{C}$);
    $\mathbf{l}^0 \leftarrow$ FUSE($\mathbf{l}^0$,$\mathbf{l}^1$,$\mathcal{C}$,$\mathcal{F}$);
end
return $\mathbf{l}^0$;
```

Algorithm 1: Main function.

The function *randomSampling* randomly selects a facility out of $\mathcal{F}$. Function *betterThanOutlier* is presented in Algorithm 2.

The stop criteria suggested by Delong et al. is a temporal criteria. We decided that a criteria based on the energy of the solution would be better, as it would be more independent of the application. The algorithm should stop after n iterations without an improvement. Since the method is monotonic, the solutions' energy does not oscillate or get higher. After testing, we chose $n = \left\lceil \frac{|\mathcal{F}|}{4} \right\rceil$.

```
Data: $j_{new}$, $\mathcal{C}$
Result: $\mathbf{l}$
for $i \in \mathcal{C}$ do
    if $c_{ij_{new}} < c_{i\emptyset}$ then
        $l_i \leftarrow j_{new}$;
    else
        $l_i \leftarrow \emptyset$;
    end
end
return $\mathbf{l}$
```

Algorithm 2: betterThanOutlier function.

4.3 Transformation of the Fusion Problem Into a minWVC Problem

To implement the Fusion procedure (here represented as the *FUSE* function) Delong et al. present the problem as a minWVC minimization. Their idea is to create a graph $G(\mathcal{V}, \mathcal{E}, w)$, where $\mathcal{V} = S^0 \cup S^1$ and then determine the vertex cover with the smallest sum of weights (recall Sec. 2.2 about minWVC problem). As $\mathcal{V} = S^0 \cup S^1$, the vertices correspond to facilities and the minimum cover $\mathcal{V}_C$ will be the set of facilities that minimizes Equation 4.3 ($S' = \mathcal{V}_C$). They demonstrate the equivalence between problems in the following theorem.

Theorem 1 *Given labellings $\mathbf{l}^0 \in \mathbf{l}(S^0)$ and $\mathbf{l}^1 \in \mathbf{l}(S^1)$, there exists an instance of minimum-weighted vertex cover $(\mathcal{V}, \mathcal{E}, w)$ with $\mathcal{V} = S^0 \cup S^1$ such that $S^* \subseteq \mathcal{V}$ is an optimal cover if and only if S^* is an optimum of equation 4.3.*

They also provide a construction for the graph $(\mathcal{V}, \mathcal{E}, w)$. The vertices are the opened facilities of the initial labellings, $\mathcal{V} = S^0 \cup S^1$. The edges in the graph are built in the following way: if a certain client i is assigned to facility j in labelling $\mathbf{l}^0$ and to facility j' in $\mathbf{l}^1$, then there must be an edge between j and j'.

For each edge, at least one of the vertices it connects must be in the cover. In the context of the UFL problem, edges connect the label that each client is assigned in $\mathbf{l}^0$ to its label in $\mathbf{l}^1$. By assuring that at least one of those vertices is in the cover, one of the two labels will be in S'. And the fusion constraint (Eq. 4.2) is met.

On a vertex cover problem, a boolean variable u is assigned to each vertex, taking the value 1 if such vertex is on the cover and 0 otherwise. In the UFL context, this means whether or not a facility is present on $\mathbf{l}'$ - if it remains opened after the fusion or if it is closed.

Equation 4.3 can be re-written in the following way:

$$G(\mathbf{u}) = \sum_{i \in \mathcal{C}} \min_{j \in \{l_i^0, l_i^1\}} u_j c_{ij} + \sum_{j \in \mathcal{F}} f_j u_j \tag{4.4}$$

$$\text{s.t. } u_j + u_{j'} \geq 1, \forall \{j, j'\} \in \mathcal{E} \tag{4.5}$$

$$\mathcal{E} = \{\{j, j'\} | \exists (l_i^0 = j, l_i^1 = j')\} \tag{4.6}$$

Condition 4.5 ensures that at least of l_i^0 or l_i^1 must be selected, verifying the fusion constraint.

By grouping the coefficients involving each variable, we get our final formulation, similar to Equation 2.7. After dropping the additive constant $\sum_{i \in \mathcal{C}} \max(c_{il_i^0}, c_{il_i^1})$, the function we want to minimize is:

$$\text{minimize} \sum_{j \in \mathcal{V}} w_j u_j \tag{4.7}$$

$$\text{s.t. } u_j + u_{j'} \geq 1, \forall \{j, j'\} \in \mathcal{E} \tag{4.8}$$

$$u_j \in \mathbb{B}, \forall j \in \mathcal{V} \tag{4.9}$$

where the weights w_j are given by Equation 4.10:

$$w_j = \sum_{i:l_i^0 = j} \min(0, c_{ij} - c_{il_i^1}) + \sum_{i:l_i^1 = j} \min(0, c_{ij} - c_{il_i^0}) + f_j \tag{4.10}$$

Recalling that we are solving a minimum weight problem, we are trying to make the weights reflect how good or how bad the labels are. Consider $c_{ij} - c_{il_i^1}$ which can be either ≥ 0 or < 0. In the first case, $c_{ij} \geq c_{il_i^1}$ and label j is a poorer choice than the corresponding label in $\mathbf{l}^1$, l_i^1 (recall that $l_i^0 = j$). In the second case, $c_{ij} < c_{il_i^1}$ and it is better to choose label j over l_i^1. For the $\min(0, c_{ij} - c_{il_i^1})$, the term will be 0 in the first case and negative (because $c_{ij} < c_{il_i^1}$) in the second one. Therefore w_j will be, in the worst case, equal to its opening cost (f_j) and will decrease as label j proves to be a better choice than other labels for some customers (situations where $c_{ij} < c_{il_i^1}$).

In order to better understand the construction of the graph $(\mathcal{V}, \mathcal{E}, w)$ from the labellings $\mathbf{l}^0, \mathbf{l}^1$, an example is provided in Appendix A.

4.3.1 Trivial Cases

There are two cases where, after the graph construction, a few vertices need to be removed. minWVC solvers usually do not cope with these cases and expect the graph to be free of such situations.

One is when a customer i is given the same label j in the two labellings, i.e., $l_i^0 = l_i^1 = j$. In that case, there is an edge $\{j, j\}$ on the graph, a *self-edge*. The vertex corresponding to label j must be in the cover and S' must include j. Such a vertex is then removed from the graph, as well as all edges connected to it.

The second case is when a vertex has a *negative weight*. Since we want to minimize the sum of weights, a vertex with a negative one will be surely part of the solution. Again, that label is added to S' and its corresponding vertex is removed from the graph.

There are situations where the graph is trivial enough to be solved analytically. Such situations include graphs with only two vertices and an edge connecting them. In that case, we just pick the vertex with the smallest weight. Other case is a bipartite graph with only one edge on one of the partitions. This is a very common situation with the linear facility fusion scheme (Figure 4.1).

4.3.2 FUSE Algorithm

The algorithm below shows how to go from the function in Equation 4.3 to the function in Equation 4.7. For this, a solver for the minWVC must be used. That is the theme of the next section. The solution of the minWVC problem is a vertex cover $\mathcal{V}_C$. As said before, vertices correspond to facilities, so the vertices remaining in the cover are the opened facilities ($S' = \mathcal{V}_C$). With S' determined, getting the resulting labeling is a straightforward task: $l'_i = \arg\min_{j \in \{l^0_i, l^1_i\} \cap S} c_{ij}$. FUSE algorithm takes as input two initial labellings and returns a new labelling that is a "fusion" of the two inputs.

The trivial cases are evaluated at two different moments. A list $\mathcal{V}_F$ is created to store all the *forced vertices*, that need to be included in S'. *Self-edges* are evaluated when the graph is being built. This evaluation is done at this specific point in the code for efficiency reasons, since we avoid searching the list of edges for *self-edges* later on. Facilities with *self-edges* are stored in $\mathcal{V}_F$. After the graph is built all facilities with negative weights are also added to $\mathcal{V}_F$. Finally, all facilities in $\mathcal{V}_F$ and the edges incident on them are removed from the graph.

```
**Data**: $\mathbf{l}^0$,$\mathbf{l}^1$,$\mathcal{C}$,$\mathcal{F}$
**Result**: $\mathbf{l}'$
for $j \in S^0 \cup S^1$ do
    $w_j \leftarrow f_j$;
end
$\mathcal{V} \leftarrow S^0 \cup S^1$, $\mathcal{V}_F \leftarrow \{\}$, $S' \leftarrow \{\}$;
for $i \in \mathcal{C}$ do
    $j \leftarrow l_i^0, j' \leftarrow l_i^1$;
    if $j = j'$ then
        $\mathcal{V}_F \leftarrow \mathcal{V}_F \cup \{\{j\}\}$ ;
    else
        $\Delta \leftarrow (c_{ij'} - c_{ij})$;
        if $\Delta > 0$ then
            $w_j \leftarrow w_j - \Delta$;
        else
            $w_{j'} \leftarrow w_{j'} + \Delta$;
        end
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\{j, j'\}\}$;
    end
end
for $j \in \mathcal{V}$ do
    if $w_j < 0$ then
        $\mathcal{V}_F \leftarrow \mathcal{V}_F \cup \{\{j\}\}$ ;
    end
end
for $j \in \mathcal{V}_F$ do
    $\mathcal{V} \leftarrow \mathcal{V} \backslash \{\{j\}\}$;
    for $e = \{u, v\} \in \mathcal{E}$ do
        if $u = j \vee v = j$ then
            $\mathcal{E} \leftarrow \mathcal{E} \backslash \{\{u, v\}\}$;
        end
    end
    $S' \leftarrow S' \cup \{\{j\}\}$;
end
$S' \leftarrow S' \cup$ MIN-WVC($\mathcal{V}, \mathcal{E}, \mathbf{w}$);
for $i \in \mathcal{C}$ do
    $l'_i = \arg\min_{j \in \{l_i^0, l_i^1\} \cap S'} c_{ij}$;
end
return $\mathbf{l}'$;
```

Algorithm 3: FUSE algorithm.

4.4 Solving the Approximated minWVC Problem as a Minimum Cut Problem

There are several ways to solve a minWVC problem.

It is easy to find a solution to a VC with a 2-approximation. This classical approach involves a relaxation from $u_j \in \{0,1\}$ to $x_j \in [0,1]$, solving the resulting linear problem and then retrieving the original variables by making $u_j = [x_j \geq \frac{1}{2}]$.

Dinur and Safra [22] show that the **minVC!** problem is NP-hard to approximate to within any factor smaller than $10\sqrt{5} - 21 \approx 1.36067$. On the other hand, VC problems on bipartite graphs can be solved converting them in max-flow/min-cut problems, which have polynomial time solvers that return exact solutions, so this is the approach we took.

In order to have a faster solver, we need to ensure that $G(\mathcal{V}, \mathcal{E}, w)$ is bipartite. This is equivalent to ensuring that S^0 and S^1 are disjoint. However, there is no warranty given by the graph construction in FUSE algorithm that S^0 and S^1 will be disjoint. If $\mathbf{l}^0$ and $\mathbf{l}^1$ share a label, than $S^0 \cap S^1 \neq \{\}$ and the graph will not be bipartite.

In order to create a bipartite graph, we build an alternative version of $G(\mathcal{V}, \mathcal{E}, w)$, $\tilde{G}(\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, w)$, and solve an approximated and over-penalized problem. In this problem, each label $j \in S^0$ is associated with a vertex represented by the boolean variable $z_j^0 \in \mathbb{B}$ and each label $j' \in S^1$ is associated with variable $z_{j'}^1 \in \mathbb{B}$. If a label j is present in the two labellings, it is associated with two vertices in $\tilde{G}$ and represented by two different variables z_j^0 and z_j^1, both $\in \mathbb{B}$.

The relation between u_j, z_j^0 and z_j^1 is simple: $u_j = 1$ if either z_j^0 or z_j^1 are 1, and it is 0 otherwise. Formally,

$$u_j = \begin{cases} z_j^0 & \text{if } j \in S^0 \backslash S^1 \\ z_j^1 & \text{if } j \in S^1 \backslash S^0 \\ z_j^0 + z_j^1 - z_j^0 z_j^1 & \text{if } j \in S^0 \cap S^1 \end{cases} \tag{4.11}$$

Our problem then becomes

$$\tilde{G}(\mathbf{z}) = \sum_{j \in S^0} w_j z_j^0 + \sum_{j \in S^1} w_j z_j^1 \tag{4.12}$$

$$z_j + z_{j'} \geq 1, \forall (j, j') \in \tilde{\mathcal{E}} \tag{4.13}$$

where $\tilde{\mathcal{E}} = \{(j, j') | \exists (l_i^0 = j, l_i^1 = j')\}$. Notice that now the edges are oriented. This implies a change in the algorithm: where we had $\mathcal{E} = \mathcal{E} \cup \{\{j, j'\}\}$, we now must make $\tilde{\mathcal{E}} = \tilde{\mathcal{E}} \cup \{(j, j')\}$.

Since we add an extra variable for each facility in $S^0 \cap S^1$, we increase the cost of $\tilde{G}(\mathbf{z})$. In the worst case, $S^0 \cap S^1 = S^0 \cup S^1$ and the cost doubles. So the bipartite minWVC problem is a 2-approximation of the minWVC problem. However, if $S^0 \cap S^1 = \{\}$, then the problem is the same and there is no approximation.

4.4.1 The Bipartite minWVC Problem

If a graph $\tilde{G}(\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, w)$ is bipartite, then the minWVC problem can be converted into a *s-t-minimum cut* problem. This conversion is done in two steps:

- We first transform the restrictions into terms in the function to be minimized so that these restrictions are always verified.
- We then convert the function to be minimized into a flow network (recall Sec. 2.1.4) and solve it by calculating its min cut.

Converting restrictions into minimization terms

If we recall the minWVC problem, we have just one restriction associated with it, Constraint 4.13. We convert such constraint into a term with a huge penalty associated to it, $\eta \bar{z_j} \bar{z_{j'}}$, where η is ideally infinite. This ensures that at least one of the two coefficients will be 1, otherwise the cost function is very large.

Recalling Equation 4.12, we can now substitute the Constraint 4.13 by the $\eta \bar{z_j} \bar{z_{j'}}$ term:

$$\tilde{G}(\mathbf{z}) = \sum_{j \in S^0} w_j z_j + \sum_{j \in S^1} w_j z_j + \sum_{(j,j') \in \tilde{\mathcal{E}}} \eta \bar{z_j} \bar{z_{j'}} \tag{4.14}$$

Converting $\tilde{G}(\mathbf{z})$ into a *s-t-minimum cut* problem

Kolmogorov et al. [8] presented a work on how PB functions (recall Sec. 3.3) like $\tilde{G}(\mathbf{z})$ can be converted into a *s-t-minimum cut* problem. They show how to build a flow network corresponding to a PB function and then determine the function's minimum by searching for the graph's minimum cut. Their only constraint is that $\tilde{G}(\mathbf{z})$ is *regular*.

A quadratic PB function is *regular* if and only if all its second order terms (represented as $E(x, y) \in \mathbb{R}$) verify the following condition:

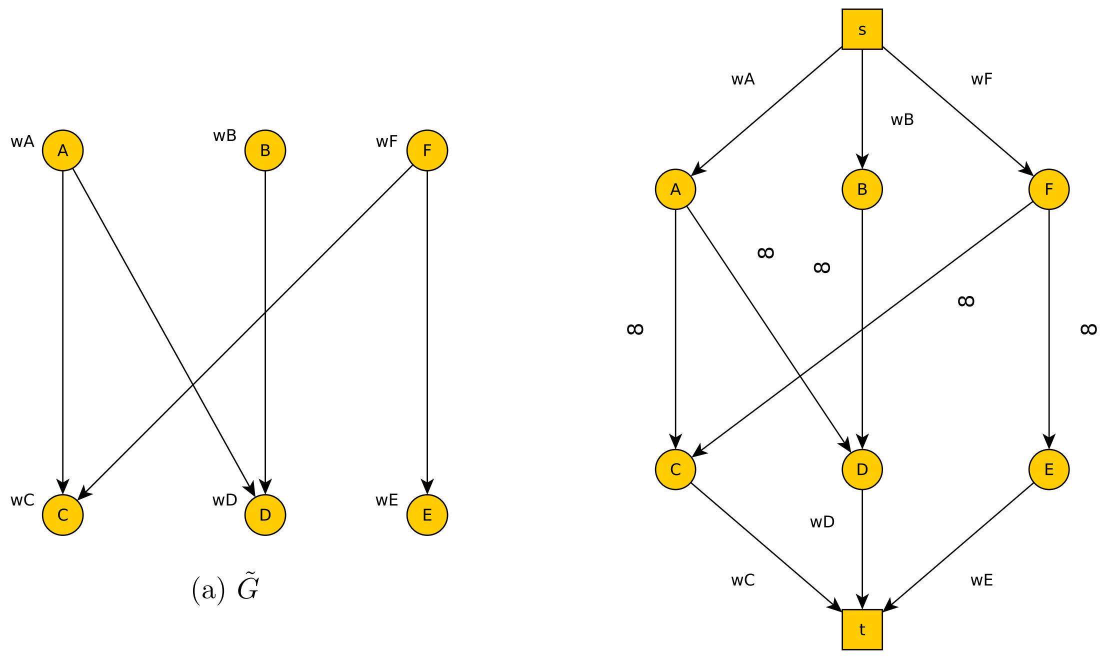
$$E(0,0) + E(1,1) \leq E(1,0) + E(0,1) \tag{4.15}$$

On our case, $\tilde{G}(\mathbf{z})$ is not regular because of the second order term $\eta \bar{z_j} \bar{z_{j'}}$: $E(0,0) = \eta \times 1 \times 1 = \eta$, $E(1,0) = \eta \times 0 \times 1 = \eta \times 1 \times 0 = E(0,1) = 0$ and $E(1,1) = \eta \times 0 \times 0 = 0$, so $E(0,0) + E(1,1) = \eta > 0 = E(1,0) + E(0,1)$, which violates Condition 4.15.

In order to turn $\tilde{G}(\mathbf{z})$ into a regular function, we need to change this term. To do that, we invert the meaning of one of the variables. In practice, what we do is invert one group of variables. Let's say we invert all variables related to S^1. $\tilde{G}(\mathbf{z})$ becomes

$$\tilde{G}(\mathbf{z}) = \sum_{j \in S^0} w_j z_j + \sum_{j \in S^1} w_j \bar{z_j} + \sum_{(j,j') \in \tilde{\mathcal{E}}} \eta \bar{z_j} z_{j'} \tag{4.16}$$

and it is now regular.



(a) $\tilde{G}$

(b) G_F

Figure 4.2: Transformation of $\tilde{G}$ to G_F.

To maintain a consistent problem, we need to update the values of the variables that are dependent on the variables related to S^1. So, $u_j = 1$ if $z_j^0 = 1$ or if $z_j^1 = 0$. More precisely,

$$u_j = \begin{cases} z_j^0 & \text{if } j \in S^0 \backslash S^1 \\ 1 - z_j^1 & \text{if } j \in S^1 \backslash S^0 \\ 1 - z_j^1 + z_j^0 z_j^1 & \text{if } j \in S^0 \cap S^1 \end{cases} \tag{4.17}$$

In order to convert $\tilde{G}(\mathbf{u})$ into a *s-t-minimum cut* problem, we follow Kolmogorov's construction. This new graph G_F is a flow network (as described in Section 2.1.4), so all edges have a capacity associated to them and the vertices no longer have weights.

Our graph G_F is built in the following fashion:

- Each term $w_j z_j$ is converted into an edge (s, j), with capacity w_j.
- Each term $w_j \hat{z}_j$ is converted into an edge (j, t), with capacity w_j.
- Each term $\eta \bar{z}_j z_{j'}$ is converted into an edge (j, j'), with capacity η (so, this edges will have ideally infinite capacity).

Figure 4.2 shows an example of the transformations from a graph $\tilde{G}(\mathcal{V}, \mathcal{E}, w)$ to the graph that represents the *s-t-minimum cut* problem, G_F. Notice how the heights form the minWVC problem become the capacities of the edges linking those vertices to the source and the sink on the *s-t-minimum cut* problem. All the previous edges now have infinite capacity.

4.5 The MIN-WVC Algorithm

As described in Section 2.3, calculating a graph's minimum cut can be done by calculating its maximum flow. That is the approach we use here: calculating G_F max flow.

In our work, we used the algorithm presented by Boykov and Kolmogorov in their work [6]. It is an algorithm developed by them while working at Siemens and later on reimplemented by Kolmogorov. Their article shows that this algorithm outperforms all others know to date. It does not work as well for more complex graphs, but that is not our case. As the graphs we are here solving are quite simple, with two levels of variables, we decided this would be an appropriate choice.

Algorithm 4 presents the pseudo-code for the *MIN-WVC* algorithm (used in Algorithm 3). This function's input is a graph $(\mathcal{V}, \mathcal{E}, \mathbf{w})$ and its output is the set of facilities S corresponding to the vertices in $\mathcal{V}$ that lie in the cover. The *maxFlow* function runs the max-flow algorithm developed by Boykov and Kolmogorov.

4.6 Back to UFL

The *minWVC* algorithm returns the set S' of opened facilities in the labelling $\mathbf{l}'$. To determine the labelling $\mathbf{l}'$, the *FUSE* algorithm makes $l'_i = \arg\min_{j \in \{l_i^0, l_i^1\} \cap S'} c_{ij}$. Remeber that, due to the fusion constraint, $\{l_i^0, l_i^1\} \cap S'$ never is an empty set.

```
Data: $\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \mathbf{w}$
Result: $S$
$\mathcal{V}_0 \leftarrow \{\}$, $\mathcal{V}_1 \leftarrow \{\}$, $\mathcal{V}' \leftarrow \{s, t\}$, $\mathcal{E}' \leftarrow \{\}$, numEdges = 0, edgesCapacities $\leftarrow \{\}$ ;
for $e = (a, b) \in \tilde{\mathcal{E}}$ do
    if $v_a^0 \notin \mathcal{V}_0$ then
        $\mathcal{V}_0 \leftarrow \mathcal{V}_0 \cup \{\{v_a^0\}\}$;
        $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{\{v_a^0\}\}$;
        $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(s, v_a^0)\}$;
        edgesCapacities[numEdges] $\leftarrow w_a$;
        numEdges $\leftarrow$ numEdges +1;
    end
    if $v_b^1 \notin \mathcal{V}_1$ then
        $\mathcal{V}_1 \leftarrow \mathcal{V}_1 \cup \{\{v_b^1\}\}$;
        $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{\{v_b^1\}\}$;
        $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(v_b^1, t)\}$;
        edgesCapacities[numEdges] $\leftarrow w_b$;
        numEdges $\leftarrow$ numEdges +1;
    end
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(v_a^0, v_b^1)\}$;
    edgesCapacities[numEdges] $\leftarrow \eta$;
    numEdges $\leftarrow$ numEdges +1;
end
$[\mathbf{z}^0, \mathbf{z}^1] \leftarrow$ maxFlow($\mathcal{V}'$,$\mathcal{E}'$,edgesCapacities);
$S \leftarrow \{\}$;
for $a \in \tilde{\mathcal{V}}$ do
    if $z_a^0 \in \mathcal{V}_0$ and $z_a^0 = 1$ then
        $u_a \leftarrow 1$;
    else if $z_a^1 \in \mathcal{V}_1$ and $z_a^1 = 0$ then
        $u_a \leftarrow 1$;
    else
        $u_a \leftarrow 0$
    end
    if $u_a = 1$ then
        $S \leftarrow S \cup \{\{a\}\}$;
    end
end
return $S$;
```

Algorithm 4: MIN-WVC algorithm.

Chapter 5

Applications in Computer Vision Problems

Our experiments are intended to determine if the FFM algorithm is an alternative to the MP algorithm and what tradeoffs it offers.

We chose two CV problems that use the MP algorithm: the Piecewise Planar Reconstruction work from Raposo et al. [2] in Section 5.1 and the work in Unsupervised Camera Calibration from Melo et al. [3] in Section 5.2 to validate the FFM algorithm.

5.1 Case 1: Application to Piecewise Planar Reconstruction

5.1.1 Introduction

Raposo et al.[2] proposes a pipeline for Piecewise-Planar Reconstruction (PPR). Given a video captured by a stereo camera, it simultaneously estimates the camera motion and reconstructs the planes in the scene. In this case, an algorithm from Michel et al. [23] is used to reconstruct the contours of the intersection of a set of virtual planes with the scene (the profile cuts).

To describe the scene, Raposo et al. need to determine the planes on it. To do so, they create a set of plane hypotheses. Creating these hypotheses involves segmenting the profile cuts into lines and combining pairs of non-skewed lines in different profile cuts.

Since some of these hypotheses are degenerate, many times due to inaccuracy in the profile cut estimation, the set of hypotheses needs to be trimmed. This can be casted as a UFL problem. Speeding up this process is crucial to have real time scene reconstruction, which is the motivation to apply the FFM algorithm in the case study.

Through plane alignment among consecutive frames, they are able to extract the camera motion. Furthermore, planes that are not well represented on a certain camera pose can be correctly labeled later on, based on images from other poses where that plane is well visible.

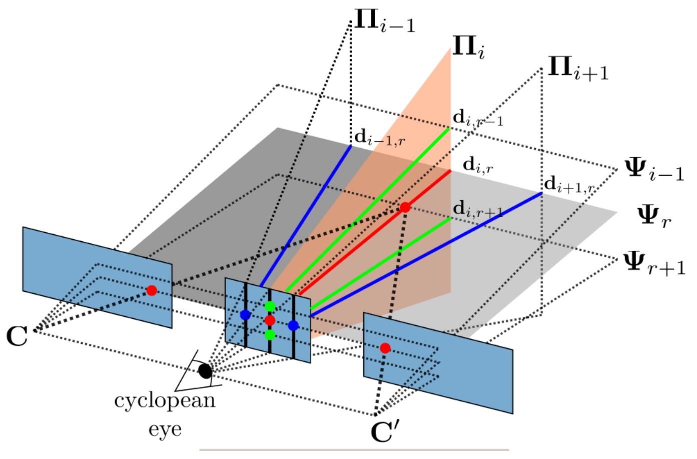


Figure 5.1: Cyclopean Eye. Each virtual plane $\mathbf{\Pi}$ intersects the scene. All backprojection rays $\mathbf{d}$ intersect at the cyclopean eye. The points along each backprojection ray $\mathbf{d}$ have an energy associated that describes the likelihood of being in the scene's structure. Source: [24]

UFL application

The UFL is used to assign pixels in the cyclopean eye to the plan hypothesis, discarding bad hypotheses because of their opening cost.

The assignment process works as follows: N virtual planes, all perpendicular to the baseline and passing through its middle point (point O) intersect the scene. For each virtual plane, an energy image is generated that represents the likelihood of the pixels in each M lines of the image being at a certain depth.

Stacking this N energy images (one for each virtual plane), we get a depth volume, with a vertex centered in the middle point of the baseline (point O, the cyclopean eye). This volume corresponds to M by N backprojections that pass through O, where each point in the backprojection ray has a likelihood of belonging to the structure in the scene.

Each backprojection is associated with a plane hypothesis. The plane whose intersection point with the backprojection maximizes the energy is selected.

The UFL is used for this assignment. Customers are the backprojection rays and facilities are the plane hypotheses. The cost c_{ij} of assigning backprojection rays i to plane hypothesis j is the energy associated to the point where plane j intersects the ray i. We aim to determine the minimal set of plane hypotheses that describe well the scene. The elements of the set of opened facilities of the final solution are considered the best hypothesis and are used in following steps of the algorithm.

Goal

More than the correct assignment between clients and customers, the main goal is to trim the set of facilities to the best plane hypotheses. The nature of the algorithm dictates that the set of opened facilities must be small, as too many plane hypothesises would make the subsequent steps of the algorithm too heavy, computationally speaking. We want the right facilities on the opened facilities set, but not too many of them.

5.1.2 Experimental Method

Our dataset is composed of 14 pairs of rectified images, from our lab's datasets. These images have 1024x768 pixels and represent indoor and outdoor scenes, captured by a Bumblebee camera. A full description of the dataset is presented in Appendix C.

For each image, we tested three algorithms: the MP algorithm, the FFM algorithm, and a hybrid algorithm, combining MP and FFM, called MPF. This hybrid algorithm is described in Section 6.2.2.

For each combination of images and algorithms, we performed 50 tests. To obtain the Groundtruth (GT) for the scenes reconstruction, we used an heavier algorithm [23], that uses not UFL but the energy-based multi-model fitting algorithm PEARL, which renders reconstructions really close to reality but at a much higher computational cost.

Evaluated Parameters

The parameters used can be separated into two sets: those which use the GT solution for comparison and those who don't.

The set of parameters that do not use the GT solution are the following:

- **NPlanes** - Number of chosen plane hypotheses (which are represented by the opened facilities in the UFL solution).
- **Energy** - Energy of the UFL solution.
- **T Exec** - Execution time, in seconds.

For the second set of parameters, we need to use the concept of *closest plane to the GT*.

We measure the angular deviation and the norm distance between each plane of the GT and all the planes selected by the algorithm. Then, we build a dual space, where the dimensions are the norm distance and the angular deviation of each plane to a given GT plane. The *closest plane to the GT* plane is the one with the smallest euclidean distance to the origin in this dual space. As each plane in the GT has one closest plane, the set of planes in the GT has a set of corresponding closest planes.

- **Ave. Ang. Dev** - Average Angular Deviation of each GT plane to its closest plane.
- **Ave. h** - Average of the errors in the norm value between each GT plane and its closest plane.
- **Attrb.** (Attribution) - Average percentage of labels attributed to the set of closest planes.
- **Attrb. R.** (Attribution Relaxed) - Average percentage of labels attributed to the set of closest planes and to those with a angular deviation to the closest planes smaller than 5°.

We used the **Attribution** and **Attribution Relaxed** parameters to perform a qualitative evaluation of the labelling provided by the 3 methods. This gives us an analysis on how strong the selected plane hypothesis is, since we suppose that if the selected planes are good, then a lot of rays (clients) will be assigned to them.

We introduced the **Attribution Relaxed** parameter in order to be able to compare algorithms that differ in their average **NPlanes** value. An algorithm that selects more planes (higher **NPlanes**) leads to a situation where each plane has less pixels assigned to it. On the other hand, the extra planes might refer to more planes close to the GT. We decided to introduce this extra parameter, that measures not only how many pixels were attributed to the plane lying closer to the groundtruth, but also its neighbouring planes. In that way, we hope to get similar results in this parameter for all algorithms, independently of the number of planes they chose.

5.1.3 Experimental Results

The tables in Appendix D present the average of the parameters for the 50 runs, in images of the dataset presented in Section 5.1.2.

The boxplots in the next section present the values for each parameter normalized by the values obtained in the MP algorithm. For each run, we divide the values obtained in FFM and MPF by the value of MP, essentially normalizing the values and using the MP algorithm as control.

We chose this approach because the variance of the values in between inputs is high, but all three algorithms show the same tendencies. This is portrayed in Figure 5.2, where the average value of **NPlanes** is presented for all 14 images, among the 3 algorithms.

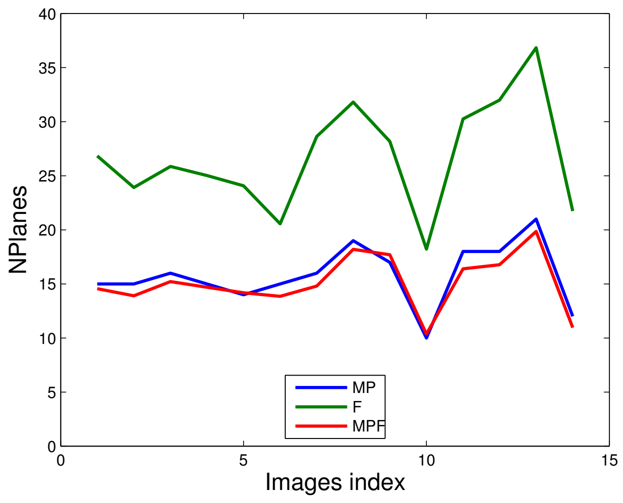


Figure 5.2: Average **NPlanes** value among the dataset.

The figure clearly shows that all three algorithms present similar behaviour. The correlation coefficient for the **NPlanes** samples of the FFM and the MP algorithms is 0.86. This reinforces the idea that the parameters can change among images, but they change in the same fashion for all three algorithms.

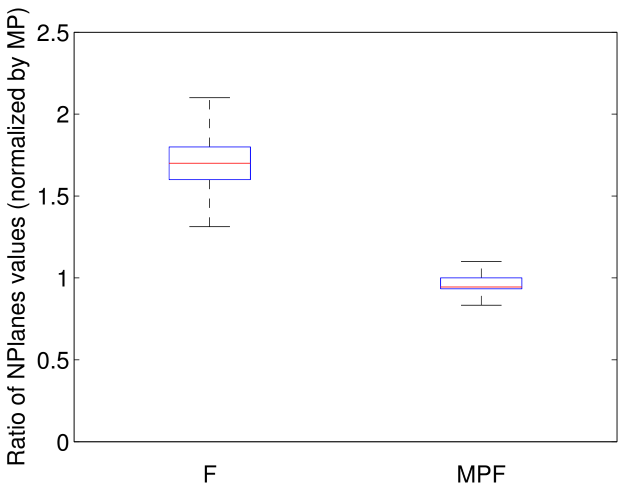
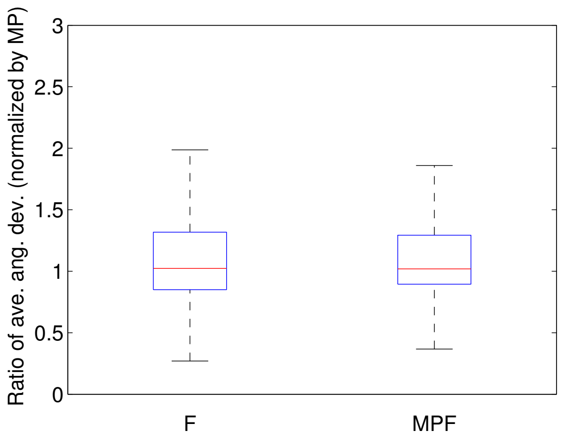
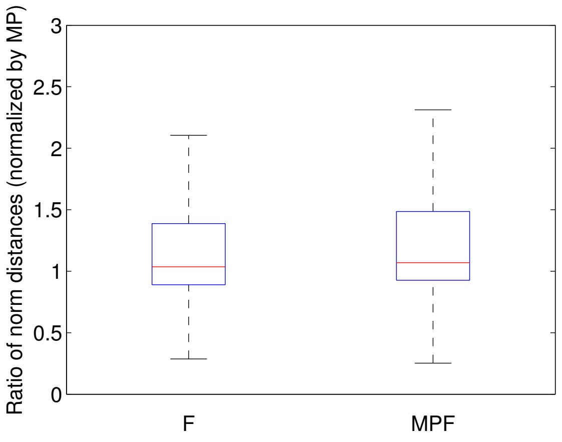


Figure 5.3: **NPlanes** ratio against the values obtained by the MP algorithm.



(a) Average Angular Deviation



(b) Average norm distance

Figure 5.4: Parameters concerning the error between GT planes and their closest planes in the algorithms' solutions.

5.1.4 Results Analysis

Figure 5.3 shows that the FFM algorithm usually selects 70% more hypotheses than the others, presenting an undesired behaviour. As said in Section 5.1.1, we want the right facilities on the opened facilities set, but not too many of them, so the FFM algorithm is not behaving as desired.

From the **Average Angular Deviation** and the **Averge h** parameters, presented in Figures 5.4a and 5.4b, respectively, we see that the 3 algorithms present similar values. This tells us that the quality of the selected hypothesis is good. The FFM algorithm is picking good hypotheses; the problem is that it is picking too many of them, with, on average, 70% more than the other two. A consequence of a bigger number of opened facilities is a higher energy parameter. In Figure 5.5, FFM algorithm has on average, an energy value 5% higher than MP and MPF.

On the other hand, when we look in terms of temporal performance (Figure 5.6), algorithms FFM and MPF present similar values, being up to 10× faster than MP.

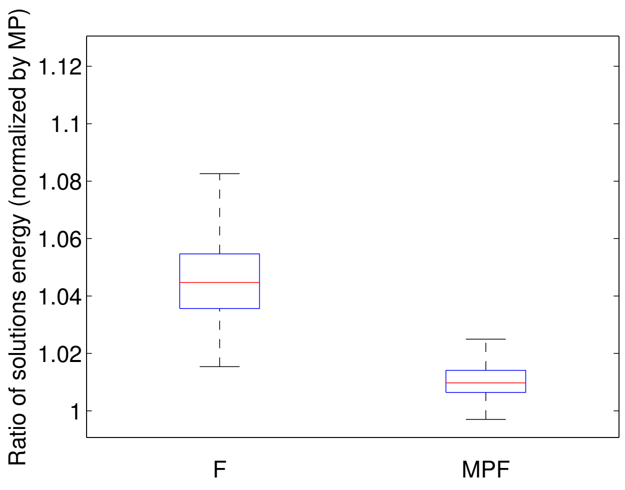


Figure 5.5: **Energy** parmeter.

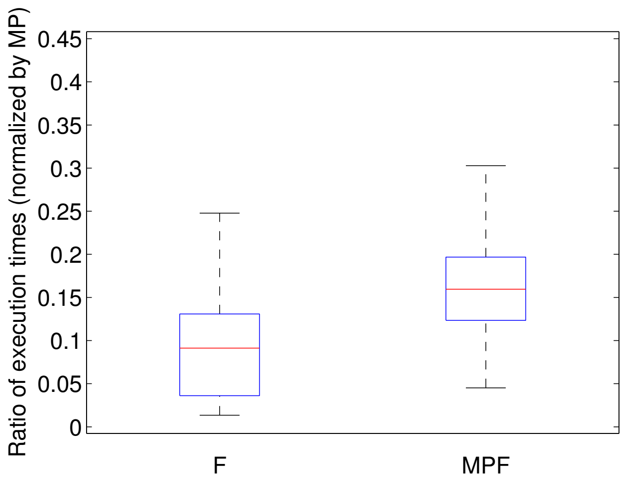


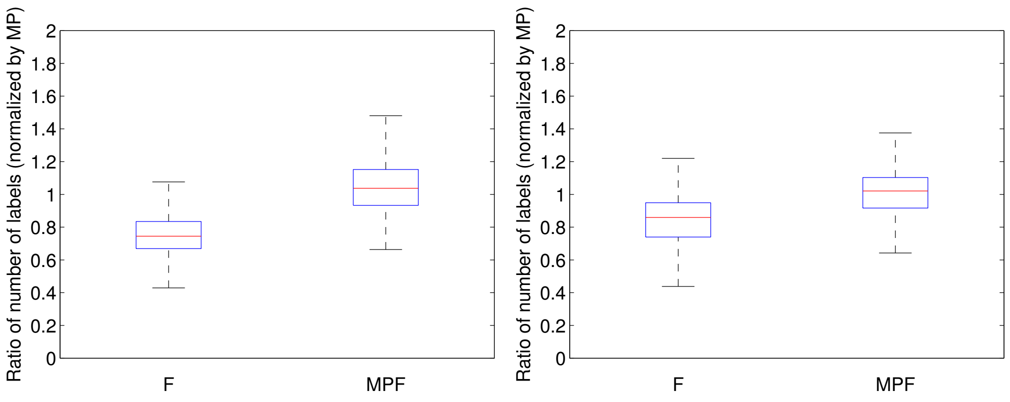
Figure 5.6: **T Exec** parameter.

Finally, analysing the attribution percentages (Figures 5.7a and 5.7b), we see that, on the non-Relaxed attribution the FFM algorithm assigns less customers to the best facilities. The higher value of **NPlanes** chosen by FFM explains this: with more planes to describe the scene, it is expected that some pixels might have another plane closer to them. This other plane might be incorrect, but locally it might be closer to a particular pixel than others.

When we analyse the relaxed version (Figure 5.7b), we see that the gap between the FFM algorithm and the others is shorter, which reinforces our idea that there might be other planes, close to the best ones, that 'steal' some pixels. The FFM algorithm still presents a worse result than the others, with 10% less pixels labelled with the best planes than MP. We believe that this has to do with the slight poorer quality of the solutions similar to what we saw in the **Energy** parameter.

5.1.5 Conclusions

Figures 5.5 and 5.3 evidence the dispersion of the values among FFM runs. The standard deviation values of the parameters in the tables in Appendix D are the highest for the FFM



(a) Attribution (b) Attribution Relaxed

Figure 5.7: Atribution parameters.

algorithm. This reflects the variable behaviour of the algorithm, which can be an issue at times. It is one of the disadvantages of the FFM algorithm when compared with the MP. This is due to the inherent randomness of the FFM algorithm, as the labels are fused in a random order. Since the solutions given by FFM are highly dependent on the first steps of fusion, if bad facilities are picked up first to be fused, it compromises the solution of that run. Such aspect is referred by Delong et al. in their paper.

We conclude that MPF is the best algorithm. It has the same quality level as MP, with the same number of opened facilities and it is as fast as FFM. Also, FFM has the highest variability.

5.2 Case 2: Application to Unsupervised Intrinsic Calibration

5.2.1 Introduction

This second case study arises from the work of Melo et al. [3]. In this work, they propose a geometric method for calibrating cameras with radial distortion, the Unsupervised Intrinsic Calibration (UIC). The method is unsupervised meaning that conics corresponding to line projections are automatically detected from low-level contour detection. The calibration parameters estimated are distortion, principal point, aspect ratio, and skew.

UIC works as follows: first, low-level image processing is applied to detect arc contours to which conics are fitted. Second, calibration hypothesis are posed by randomly combining triplets of conics (usually 50 hypothesis are created). From this set of hypotheses, we need to determine the best one. That is where the UFL is used.

Sometimes lines in 3D are fragmented in multiple arcs belonging to slightly different conics curves. Other times, as the contour detector looks for arcs, actual curves in the scene are mistaken for lines. This mistakes influence the calibration hypothesis posed.

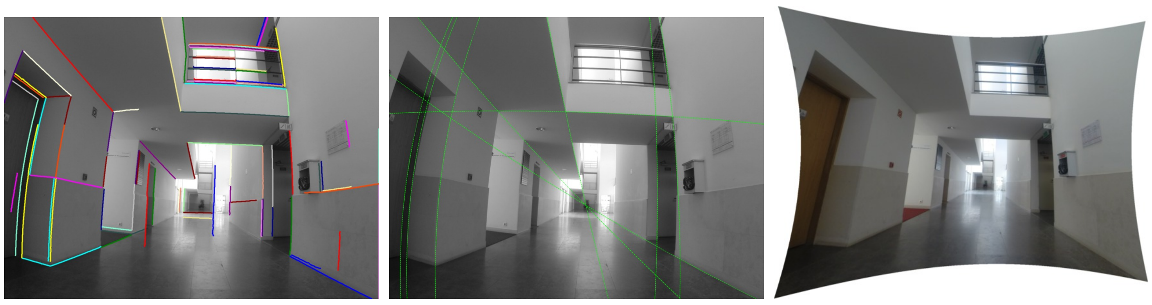

(a) Detected segments on the image. (b) Re-estimated conics. (c) Corrected image.

Figure 5.8: Image 16 during and after UIC.

UFL problem

A UFL instance is associated with each calibration hypothesis. For each one, the conics fitting the contours are re-estimated taking into account the camera parameters in order to obtain a geometrically consistent set of line projections. A good calibration model will assign better conics that diminish the distance between their points and the points in the segment.

The set of customers is the set of detected segments in the image; and the set of facilities are the re-estimated conics. The minimization of the energy of each UFL instance favours the clustering of segments that were fragmented and penalizes bad hypotheses that re-estimate the conics far from the segments. The calibration hypothesis with lowest UFL energy is selected as the chosen calibration. The distance c_{ij} between customers and facilities is given by the distance from the points in a segment i to the points in the conic j. All the facilities have the same opening cost f_j. Figure 5.8a shows the segments detected (the clients) in an image from the dataset and Figure 5.8b shows some of the re-estimated conics (the facilities). Figure 5.8c presents the output of the algorithm: the initial image without distortion.

The temporal gains in this method could be used to test more hypotheses or to improve the contour detector. Usually, when the algorithm fails to detect the calibration of a camera, it is due to poor detection of the segments, which prevent the algorithm of posing the right hypothesis. Therefore, temporal gains could be used in a more complex or high-level contour detection algorithm.

5.2.2 Experimental Method

The experiments conducted evaluate the performance in terms of quality and execution time. Our dataset consists of 17 images, obtained by 3 different cameras, with 6 different calibrations in total. They represent indoor and outdoor scenes, with and without occlusions. Full descriptions are in Appendix C. The GT for the calibration models was obtained with [25].

For each image in the dataset, we run 10 tests for the MP algorithm and the FFM algorithm.

Goals

For each instance, we assume that the algorithms should be able to associate the lowest energy across all models with the calibration model closest to the GT. This might not always be the case, not because the algorithm failed, but due to the poor quality of the image or of the segments detected.

We have seen situations where the algorithms reject the GT in favour of another calibration. This happens when there is noise in the image, when the detected segments are not clear enough, or when elements in the image which are not segments are detected as so (e.g, curves in the scene). In those situations, another calibration model might fit the detected elements better than the groundtruth. The best fitting model may not always be the correct choice.

When segments with good quality are detected, a poor choice might still be made if the calibration hypotheses derived from those segments are not good. This happens due to the combinatorial nature of the method used to create those hypotheses. Even if the best hypothesis is the chosen one, the errors might still be high because they are measured as the distance to the groundtruth and we cannot assure that the best hypothesis will necessarily be close to the groundtruth.

The connectivity test

In order to be able to evaluate the performance of the algorithms, regardless of the quality of the hypotheses, we devised another test. Consider two (or more) segments that lie in the same line in the scene (co-linear segments). For good calibration models, they should be clustered, i.e., given the same label. This test is based in the principle that good models will give the same label to two segments that lie in the same line. Therefore, if the algorithm is good, it should cluster segments that are co-linear. The test consists of checking if, in the labelling correspondent to the winning hypothesis, the algorithms give the same label to co-linear segments.

We manually chose a set of co-linear segments from a series of images and then measured how many of them were attributed the same label. An example of a set of segments used in the connectivity test is shown in figure 5.9. Segments might also be labeled as outliers. If both algorithms do so, this is neither a success or a failure. It might be due to other causes. We think that an algorithm has a poorer performance if it fails to give the same label to all segments, when the other algorithm was able to do it.

Evaluated Parameters

The parameters are the following:

- **Dist2GT** - Relative error of the calibration parameters between the chosen calibration hypothesis and the GT.
- **T Exec** - Execution time, in seconds.

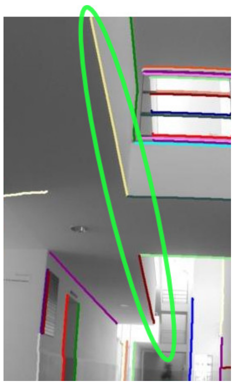


Figure 5.9: Detail of two colinear segments, from Image 16 (in Figure 5.8a).

- **Energy** - Energy of the winning solution.
- **Convergence** - Percentage of runs where the two algorithms agree on the best solution.

Recalling that the estimated parameters in the calibration hypothesis are the distortion, principal point, aspect ratio, and skew, in this experiment we assume that the aspect ratio is always 1 and the skew is always 0 because modern cameras present those properties. The distortion is represented by the η (eta) parameter. The principle point in the images is represented by (c_x, c_y).

For the connectivity test, we evaluate the percentage of co-linear segments that are given the same label, also taking into account how many of them are given the outlier label.

5.2.3 Experimental Results

The average and standard deviation values among the 10 runs are present for the parameters **Dist2GT**, **T Exec** and **Energy** in Appendix F (in the form *average* ± *standard deviation*). The **convergence** parameter is the percentage of runs, among the 10, where the two algorithms pick the same solution.

The results of the connectivity test are presented in table 5.1 below.

Image Index	Algorithm	Connected segs	Outliered Segs	Other
3	MP	4	1	0
	F	3.7	1.3	0
5	MP	1	0	2
	F	1	0	2
11	MP	1	4	0
	F	0.7	4.3	0
16	MP	1	2	0
	F	1.8	1.2	0
17	MP	2	2	2
	F	0.8	4.4	0.8

Table 5.1: Results of the connectivity test.

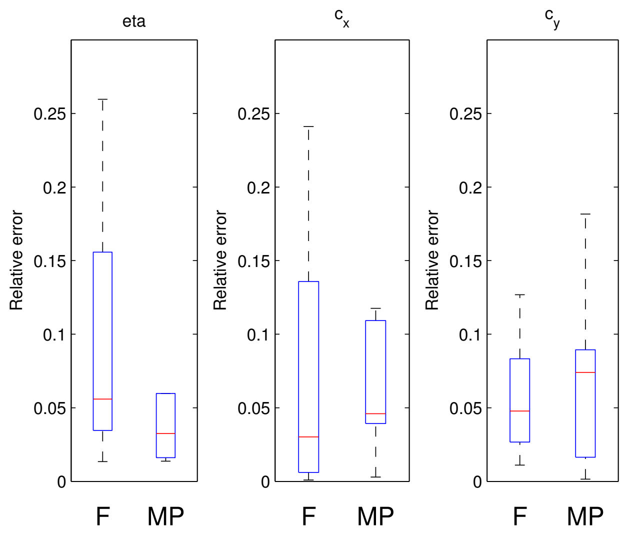


Figure 5.10: Relative error in the 3 calibration parameters (η, c_x and c_y) when the algorithms disagree.

5.2.4 Results Analysis

First off, we analyse how often the two algorithms agree. This means that, for both of them, the model with the lowest energy is the same. Table 5.2 shows the percentage of runs where both algorithms agree, among the dataset.

Image Index	1	2	3	4	5	6	7	8	9
Convergence (%)	90	100	20	100	100	100	100	100	80

Image Index	10	11	12	13	14	15	16	17
Convergence (%)	100	40	50	20	30	50	10	80

Table 5.2: Percentage of runs where both algorithms converge to the same calibration solution.

Most of the time, the two algorithms agree. However, we need to analyse how they behave when they disagree. Figure 5.10 presents the relative error of the calibration parameters of the chosen hypothesis for the two algorithms, when the model with the lowest energy is not the same. The calibration parameters considered are η, c_x and c_y. Figure 5.11 presents the norm of the relative error vector $(\varepsilon_\eta, \varepsilon_{c_x}, \varepsilon_{c_y})$. The relative error ε of each parameter is obtained as $\varepsilon = \frac{|v - v_{approx}|}{|v|}$, where v is the expected value and v_{approx} is the value read. In this way, we analyse the error in the full hypothesis and not each component separately. This gives us a better insight on the quality of the solutions because an hypothesis that has a bigger error in the first parameter might have a smaller error in the remaining two. As the quality of the hypothesis is dependent on the three parameters and not just on one of them, the norm of the relative error vector is more relevant to compare the performances.

In Figure 5.10 we see that FFM tends to select calibration models with bigger error in the distortion parameter but centers closer to GT.

Figure 5.11 shows that, overall, FFM chooses worst calibration hypotheses, since the

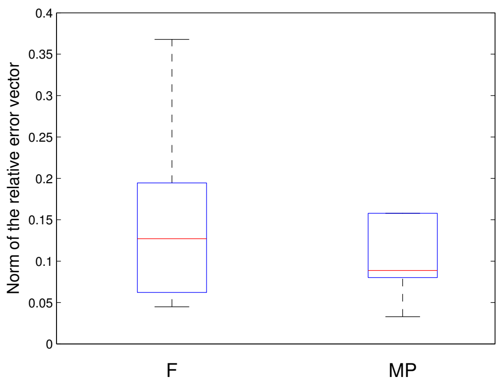


Figure 5.11: Norm of the relative error vector in the runs where the algorithms disagree.

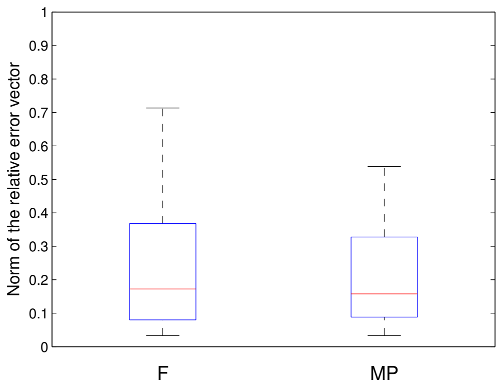


Figure 5.12: Norms of the error vectors, in all runs.

norm of the relative error vector is, on average, 50% higher. Another aspect also visible in this figure is the bigger dispersion of values for the FFM algorithm. Typically, the standard deviation of the parameters in the FFM runs is higher than the MP's.

We have seen that, when in disagreement about the best hypothesis, it is mostly FFM choosing poorer solutions. However, when they agree, the values of the norm of the relative error vector presented in Figure 5.12 show a dispersion over a larger interval.

The reason why we think we observe this behaviour is that the algorithms only disagree when there are two or more solutions close to the GT. When the solutions are further apart or when they lie farther from the GT, the two algorithms tend to agree because the overall energy of each model rises, leading to a better discrimination between them.

We have to recall that this error, which is the distance from the chosen model to the GT, is dependent of the quality of the hypothesis generated in the previous steps of the algorithm. If the hypothesis are bad, the error will be large even if the chosen model is the best one.

Concerning the time results, Figure 5.13 shows the ratio between the execution times of the FFM and MP algorithms. Each image takes a different amount of time to be processed, depending on the number of detected segments. The best way to compare the two algorithms is to use one of them as reference. The present values assume that, for a given run, the time taken by the MP algorithm is 1 and presents the values of the fusion algorithm normalized by the corresponding value. Figure 5.13 clearly states that the FFM algorithm is faster,

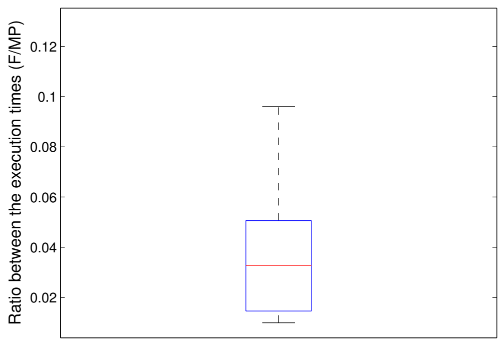


Figure 5.13: Ratio between the execution times, for all runs.

being 25 times faster on average and up to 50 times faster in the best case.

Regarding the variability of the results, a quick look at the tables in Appendix E shows that the FFM algorithm tends to provide different solutions for the same input. The disagreement on the best model between MP and FFM algorithms has to with this behaviour. Even if most of the times the fusion algorithm picks the same solution as the MP, sometimes it does not, due to this errand behaviour. The explanation has to do, as previously stated, with its random nature. The FFM algorithm is highly dependent on the fusion scheme and on the initial labellings being fused.

Connectivity test results

In Table 5.1 we have the average number of sets of segments that where given the same label (*Connected segs*), the average number of sets where all segments were labeled as outliers and the average number of remaining situations, that we consider real failure.

As seen before on the other tests, the FFM algorithm does not provide a performance as good as the MP algorithm, although it is close. Both algorithms show a good performance, since the situations of real failure (fifth column) barely exist. The FFM algorithm tends to give the outlier label to the segments more often than the MP algorithm. But, on the other hand, it has fewer situations where the co-linear segments are given different labels.

One extra note regarding the fractional values in the FFM algorithm results. While the MP selects consistently the same solution in the 10 runs, the FFM algorithm sometimes picks different solutions. This is why the MP algorithm always exhibits integer values in Table 5.1 while FFM does not.

5.2.5 Conclusion

In this application the FFM algorithm exhibits a random behaviour. It terms of quality, it can achieve, at times, a performance as good as MP. But, other times it picks poorer solutions. Overall, this random behaviour is undesirable. However, its temporal performance (on average 20$\times$ faster than MP) still makes it an interesting alternative.

Chapter 6

Algorithmic Improvements

6.1 Introduction

This final chapter is dedicated to further improvements to the FFM algorithm. These improvements are aimed at enhancing the quality of the solutions provided by the algorithm. and making the algorithm perform even faster. The next section is dedicated to mechanisms that improve the quality of the solution.

6.2 Quality Improving Techniques

6.2.1 Tree shaped fusion

In the linear fusion scheme (Fig. 4.1) the labelling $\mathbf{l}^1$ is based on just one facility. At each step, a new labelling $\mathbf{l}^1$ is built and then fused with $\mathbf{l}^0$ (recall the algorithm in Section 4.2). Instead of joining facilities one-by-one, we form a *tree shaped fusion scheme*, which fuses facilities in stages, as pictured in Figure 6.1.

In the first stage, each facility (green squares) generates a labelling (yellow circles) through the greedy *betterThanOutlier* function (purple rectangles, see algorithm in section 4.2). Then, these labellings are combined in pairs and fused (blue rectangles), creating new labellings. In the next stage, these new labellings are again combined and fused. Through successive fusions, we get one final labelling (the orange circle on top). The order of the facilities in the initial stage (and consequently, their pairing) is random.

This technique has a fixed number of fusion steps. This does not follow the original algorithm principle where as many fusion moves as wanted could be made, while also not enabling the use of temporal constraints as a stop criteria.

In order to validate this technique, we tested it over the UIC case study. We reevaluated the convergence parameter over the dataset, making 10 runs with this algorithm as in previous trials and checking whether or not this version of the algorithm converges to the same solutions as the MP algorithm. The results are shown in Table 6.1.

The tree fusion scheme in Figure 6.1 does not include a labelling $\mathbf{l}^0$ with only outlier

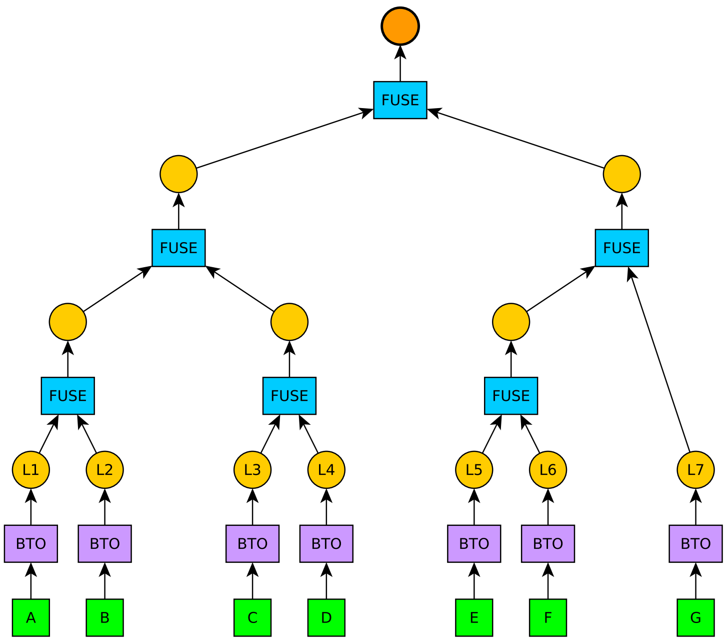
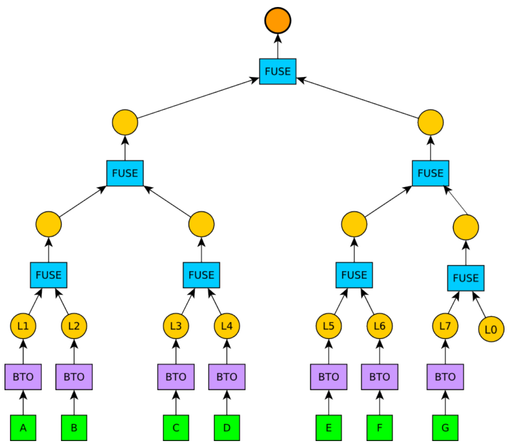


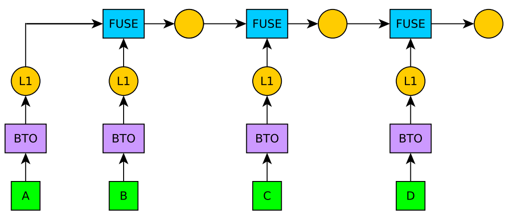
Figure 6.1: Tree fusion scheme.

labels, that is present in the linear fusion scheme. In order to evaluate the influence of this labelling, we tested two extra versions of the tree and linear scheme: *Tree_L0* which includes a labelling $\mathbf{l}^0$, and *Linear_nL0* which does not include $\mathbf{l}^0$. Figure 6.2a presents the Tree_L0 scheme and Figure 6.2b presents the Linear_nL0 scheme. The results in Table 6.1 show clearly that the presence or omission of $\mathbf{l}^0$ do not influence the convergence results.

These results show that the tree fusion scheme increases the convergence results by 38%, improving them from a convergence in 69% of the cases to 94%. This improvement is related to the fact that the tree fusion scheme does not have only one starting point, but several. Delong et al. stated that the results of each run of FFM were highly influenced by the starting point of the algorithm and claimed that FFM rarely recovers from a bad initialization. Therefore, we think that the tree fusion scheme reduces this problem.



(a) The Tree_L0 scheme.



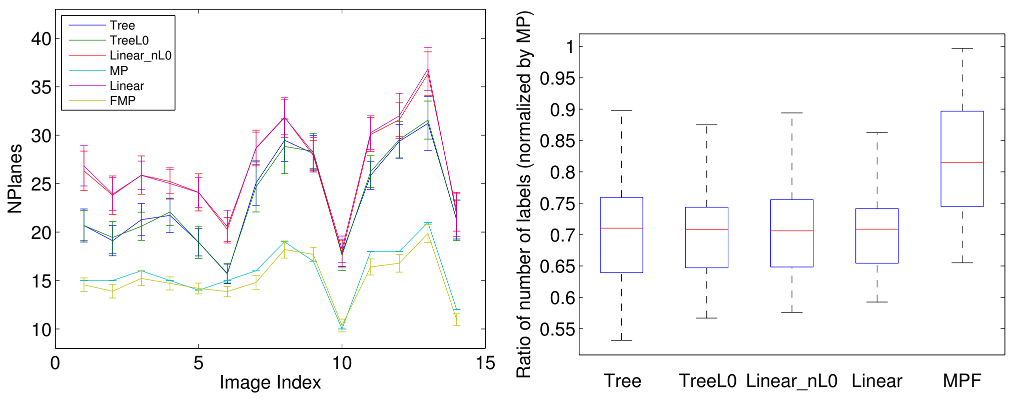
(b) The Linear_nL0 scheme.

Figure 6.2: Auxiliary fusion schemes used to evaluate the influence of the $\mathbf{l}^0$ labelling .

Image Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Linear	90 %	100 %	20 %	100 %	100 %	100 %	100 %	100 %	80 %	100 %	40 %	50 %	20 %	30 %	50 %	10 %	80 %
Tree	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	60 %	100 %	100 %	100 %	40 %	100 %	100 %	100 %	100 %
Tree_L0	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	60 %	100 %	100 %	100 %	40 %	100 %	100 %	100 %	100 %
Linear_nL0	90 %	100 %	20 %	100 %	100 %	100 %	100 %	100 %	80 %	100 %	40 %	50 %	20 %	30 %	50 %	10 %	80 %

Table 6.1: Convergence parameter for 4 different fusion schemes.

We also tested the tree fusion scheme in the PPR case study. Figure 6.3a shows the average values of the **NPlanes** parameter. The vertical bars present the standard deviation values.



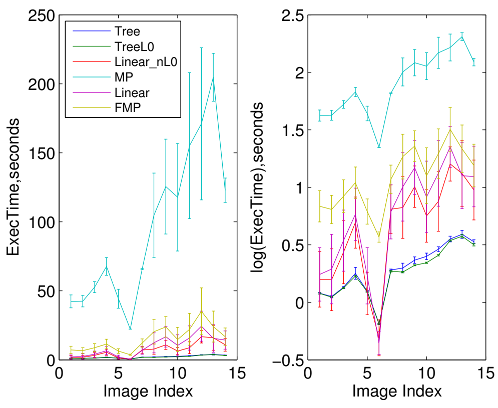
(a) **Nplanes**. (b) **Attrb. R.** normalized by MP.

Figure 6.3: Parameters for the application of the different fusion schemes to PPR case study.

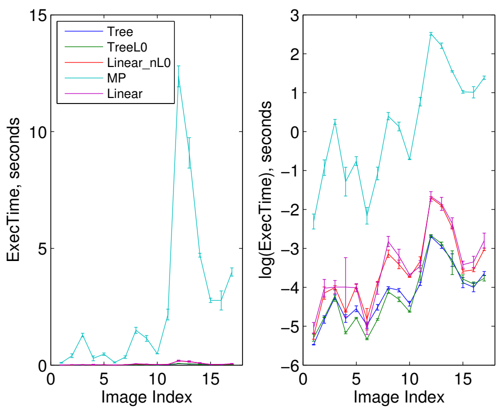
The results show that the tree scheme has a better performance than the linear scheme, since it opens less facilities. However, it does not reach the quality levels of MP, since **NPlanes** is still 25% higher than for the MP runs. Figure 6.3b presents the **Attrb. R.** parameter, normalized by the MP runs. We notice a few aspects: the tree scheme does not improve the number of labels given to the best planes, all the fusion schemes present the same average, and the methods that include the labelling $\mathbf{l}^0$ have a lower dispersion of values.

The presence of $\mathbf{l}^0$ pushes more clients into being classified as outlier. It is natural to see the top of the **Attrb. R.** distribution at a lower value. However, as those clients are discarded as outliers, the remaining rays that were assigned to the same plane of the outliered rays are now free to be assigned to other planes that did not describe the outliered rays as well, but that might be a better fit for the remaining points. This might also lead to the assignment of clients to planes closer to the GT, increasing **Attrb. R.**. Therefore, the presence of $\mathbf{l}^0$ decreases the algorithm's variability. This is confirmed by the standard deviation values, which are consistently lower for the runs that include $\mathbf{l}^0$.

In terms of temporal performance, the tree scheme outperforms all other methods tested. Figures 6.4a and 6.4b show the **T Exec** parameter for the first and second case studies, respectively. In each case study, **T Exec** is presented in its real and logarithmic domains. In both figures we see that, not only the execution times for the tree schemes are lower, but they also have a smaller standard deviation. This is expected, since this method has a fixed number of iterations, only depending on $|\mathcal{F}|$.



(a) **T Exec** in the first case study.



(b) **T Exec** in the second case study.

Figure 6.4: Temporal performance comparison for the different schemes.

Overall, the tree scheme outperforms the linear scheme in temporal and quality parameters. However, it does not lower the FFM's variability. When compared to MP, it is faster in both applications. The FFM tree scheme is a better alternative than MP for the second case study since it generally provides same quality results, but not for the first one. Also, fusion schemes that include the $\mathbf{l}^0$ labelling show lower variability (as seen in Figure 6.3b), meaning that $\mathbf{l}^0$ should be always included.

6.2.2 Hybrid algorithm

The main objective of using UFL in the PPR application is to trim the set of facilities down to a few good ones. In Figure 5.3, we see that the FFM algorithm opens, on average, 70% more facilities than the MP algorithm. However, the temporal results presented in Figure 5.6 show that the FFM is able to select these facilities 10× faster than MP. So, we thought about using the FFM as a "filter", in the sense it would trimm the set of facilities, and then run the MP algorithm over those facilities.

We considered a scenario where the FFM algorithm is run first just to filter bad facilities. Then, we create a new UFL instance, where the set of facilities is the set of opened facilities in the FFM solution. This new instance is solved by the MP algorithm, which now runs in a fraction of the original time, since the number of the facilities is smaller. We called this hybrid algorithm *MPF*, because it both MP and Fusion Moves to retrieve a solution.

Figure 5.6 shows that MPF is, on average, 6× faster than MP. Also, its results in terms of the **NPlanes** (Fig. 5.3) and **Energy** (Fig. 5.5) parameters show that MPF has a quality performance similar to MP.

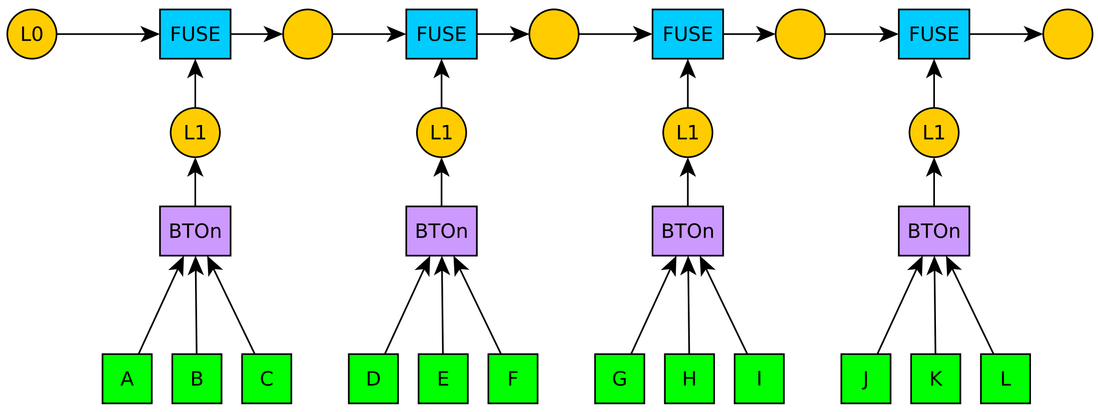


Figure 6.5: Multiple label fusion scheme, with $FPM = 3$.

So, we get a procedure where we have the same amount of opened facilities, with roughly the same quality, in a fraction of the original time. One drawback of MPF is that the variability of FFM is present, varying the set of open facilities. In applications where trimming the set of facilities is an important factor, this hybrid algorithm is a good alternative.

6.3 Processing Speed Improvement Techniques

This section presents techniques to improve the throughput performance of the algorithm.

6.3.1 Multiple facility linear fusion scheme

The idea behind this scheme is to evaluate more than one facility at each fusion move. In the linear fusion scheme (Fig. 4.1), at each step, a new labelling $\mathbf{l}^1$ is created based on a sole facility. We propose a new scheme where more than one facility is evaluated when creating $\mathbf{l}^1$. This results in fewer fusion moves to achieve convergence.

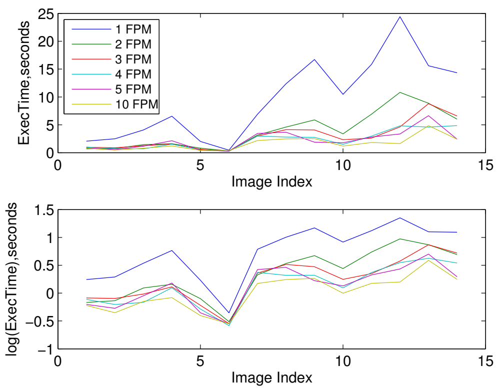
Figure 6.5 presents this new scheme. Function *betterThanOutlier_n (BTOn)* evaluates FPM facilities at a time (from the vector $\mathbf{j_{new}}$), creating a labelling $\mathbf{l}^1$ with more than just two facilities. FPM means *facilities per move*. The pseudo-code for function $BTOn$ is presented in the algorithm bellow.

```
**Data**: $\mathbf{j_{new}}$, $\mathcal{C}$
**Result**: $\mathbf{l}$
**for** $i \in \mathcal{C}$ **do**
    $l_i \leftarrow \text{argmin}_{j \in \{\mathbf{j_{new}} \cup \{\emptyset\}\}} c_{ij}$
**end**
**return** $\mathbf{l}$
```

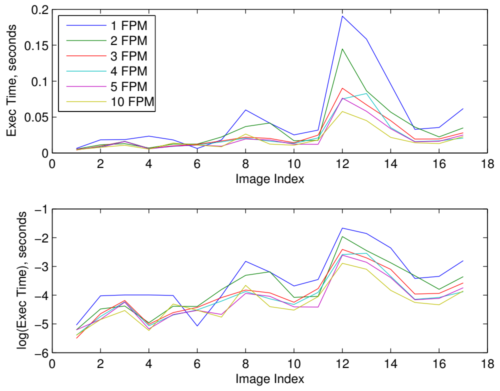
Algorithm 5: betterThanOutlier_n algorithm.

The reason for the speedup is evident: if we evaluate several labels at once, it will take us less fusion moves to evaluate all of them and to eventually reach convergence.

Figures 6.6a and 6.6b present the values of temporal performance for the PPR and UIC case studies, respectively. It is notorious, specially for images that take longer to compute, that the more facilities we fuse at a time, the faster the algorithm converges.



(a) PPR case study.



(b) UIC case study.

Figure 6.6: **T Exec** for the multiple label fusion scheme in the two case studies.

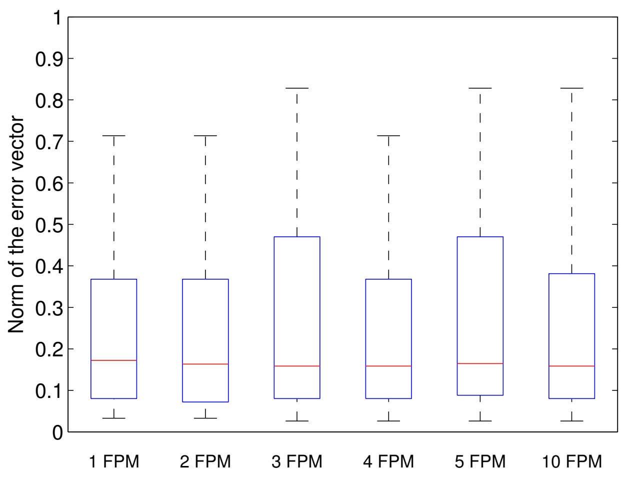
Our analysis of the quality parameters is inconclusive. Take for instances the convergence results in Table 6.2. It is not clear which version performs better. We would expect a decrease of quality, as explained bellow, but this is not what we observed. Figures 6.7a and 6.7b confirm these results. Figure 6.7a presents the norm of the calibration relative error vector for different values of FPM in the UIC case study. Figure 6.7b presents the values of **NPlanes** in the PPR case study. Again, there is no perceptible quality decrease when FPM increases.

Image Index	1	2	3	4	5	6	7	8	9
1 FPM	90 %	100 %	20 %	100 %	100 %	100 %	100 %	100 %	80 %
2 FPM	100 %	100 %	10 %	90 %	100 %	100 %	90 %	100 %	70 %
3 FPM	90 %	100 %	10 %	90 %	100 %	100 %	90 %	100 %	50 %
4 FPM	90 %	100 %	10 %	80 %	70 %	80 %	100 %	100 %	70 %
5 FPM	80 %	100 %	40 %	90 %	100 %	60 %	100 %	100 %	60 %
10 FPM	90 %	90 %	20 %	90 %	80 %	80 %	80 %	90 %	60 %

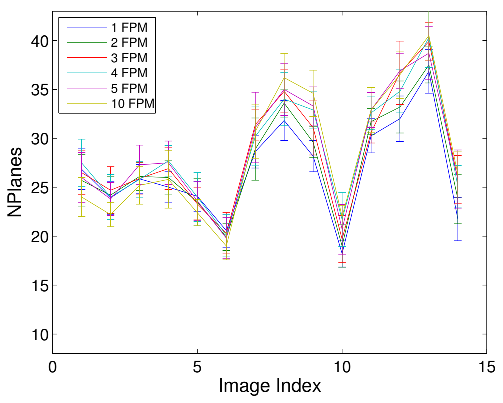
Image Index	10	11	12	13	14	15	16	17
1 FPM	100 %	40 %	50 %	20 %	30 %	50 %	10 %	80 %
2 FPM	100 %	40 %	60 %	30 %	20 %	70 %	50 %	80 %
3 FPM	90 %	30 %	70 %	50 %	0 %	60 %	20 %	90 %
4 FPM	100 %	30 %	60 %	60 %	40 %	60 %	60 %	100 %
5 FPM	100 %	30 %	70 %	10 %	30 %	30 %	80 %	90 %
10 FPM	80 %	20 %	50 %	70 %	30 %	50 %	70 %	100 %

Table 6.2: Convergence parameter in the UIC case for different values of FPM.

Consider the set of clients P_j which prefer facility j to outlier, i.e., $P_j = \{i \in \mathcal{C} : c_{ij} < ci\emptyset, j \in \mathcal{F}$. If, for two facilities $j, j' \in \mathcal{F}$, $P_j \cap P_{j'} = \{\}$, then there is no difference in result between the linear fusion scheme and the multiple label scheme, simply because they will not be competing for the same customers.



(a) Relative error vector norm for different values of FPM, in the UIC case study.



(b) **NPlanes** for different values of FPM, in the PPR case study.

Figure 6.7: Quality evaluation parameters from the two case studies.

The decrease in quality should be originated when $P_j \cap P_{j'} \neq \{\}$. This means there will be at least one customer i for which $c_{ij} < c_{ij'} < c_{i\emptyset}$ (we assume that $c_{ij} < c_{ij'}$, but it could be $c_{ij'} < c_{ij}$; it is arbitrary). Even though j' is still better than outlier, it is worse than j, so $l_i = j$. This will compromise its weight $(w_{j'})$ and result on a possible rejection from the cover in the minWVC algorithm. If it was evaluated alone, it would be picked to be the label of client i in $\mathbf{l}^1$, decreasing its weight $w_{j'}$ and making it a more interesting option.

The stop condition is still having $\left\lceil \frac{|\mathcal{F}|}{4} \right\rceil$ fusion moves without a change in the energy of the solution. So, a facility like j', which was compromised by j, will be paired with a different facility on a latter fusion move, which might not be competing for its customers.

The more facilities we evaluate at each step, (and the bigger $|\mathbf{j_{new}}|$ is), the more likely is that $\exists j, j' \in \mathbf{j_{new}}, P_j \cap P_{j'} \neq \{\}$.

Overall, this scheme improves the temporal performance of the algorithm, but its impact on quality is not clear.

Chapter 7

Conclusions

The FFM algorithm provides a mechanism to successively fuse labellings in a monotonic way to approach a good solution to the UFL problem. This mechanism involves casting the fusion problem as a minWVC problem, and then solving a 2-approximation min-cut/max-flow instance.

This mechanism provides no guarantees to achieve the optimal solution, but we show empirically that it is able to give good solutions to the initial problem. However, it is highly dependent on the fusion scheme and its random nature occasionally makes suboptimal choices.

With the two applications to practical CV cases presented in Chapter 5, we show that FFM is not a convincing alternative to the MP algorithm. It is one order of magnitude faster, but provides slightly worse solutions. We also build an hybrid version that is able to provide solutions as solid as the MP while maintaining the speed of the FFM algorithm.

In Chapter 6 we analyse other fusion schemes. This tests confirm the algorithm's high dependence on the fusion scheme used.

Overall, the FFM's linear scheme is not capable of achieving the quality of MP. Other fusion schemes that we tested might close the gap, but cannot fully replace MP. A new way to initialize the labellings needs to be found to make this algorithm truly appealing.

7.1 Future work

FFM's iterative fashion of the linear fusion scheme is an impediment to its parallelization. Each fusion move is computationally light, but it requires the previous fusion move to be complete. This creates a high data dependency between iterations.

In chapter 6 we were able to see how much the fusion scheme influences the outcome. Therefore, it might be interesting to study fusion schemes not only to see if it is possible to eradicate the algorithm's errant behaviour but also lower the data dependencies, enabling paralellization.

On another perspective, given the reduced number of works, it would be interesting to study other CV problems that could be casted as UFL problems.

Finally, studying alternatives to the 2-approximation of the min-cut/max-flow instance would also be interesting, even though we do not think that the algorithm's negative aspects are originated by this approximation.

Appendix A

Fusion Graph Construction Example

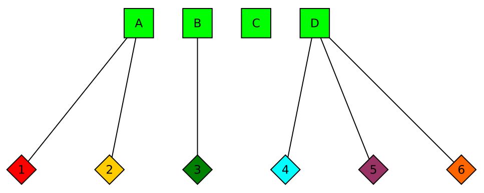
Let's consider a UFL problem with 6 clients ($\mathcal{C} = \{1, 2, 3, 4, 5, 6\}$) and four facilities/labels ($\mathcal{F} = \{A, B, C, D\}$). D is the outlier label, so D has null opening cost and constant distance to clients. The matrix of distances between clients and facilities is the following:

$$\mathbf{C} = \begin{bmatrix} 10 & 12 & 2 & 10 \\ 4 & 15 & 10 & 10 \\ 8 & 7 & 22 & 10 \\ 9 & 14 & 15 & 10 \\ 12 & 14 & 8 & 10 \\ 20 & 10 & 22 & 10 \end{bmatrix}$$

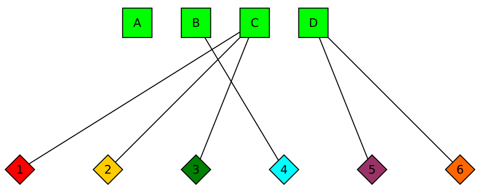
And the facilities' opening costs are:

$$\mathbf{F} = \begin{bmatrix} 18 & 19 & 19 & 0 \end{bmatrix}$$

Now, at a given point on our algorithm, we have to fuse to labellings. They are $\mathbf{l}^0 = (A, A, B, D, D, D)$ and $\mathbf{l}^1 = (C, C, C, B, D, D)$. So, $S^0 = \{A, B, D\}$ and $S^1 = \{B, C, D\}$. Figure A.1 shows the two initial labellings $\mathbf{l}^0$ and $\mathbf{l}^1$.



(a) $\mathbf{l}^0$

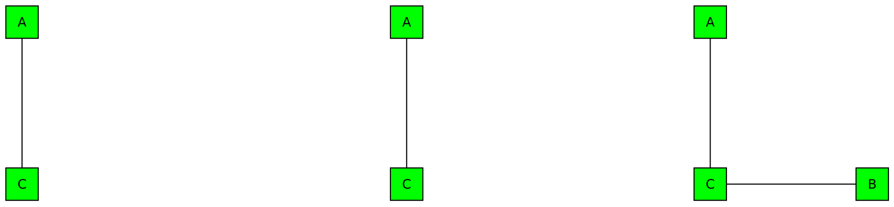


(b) $\mathbf{l}^1$

Figure A.1: Initial labellings.

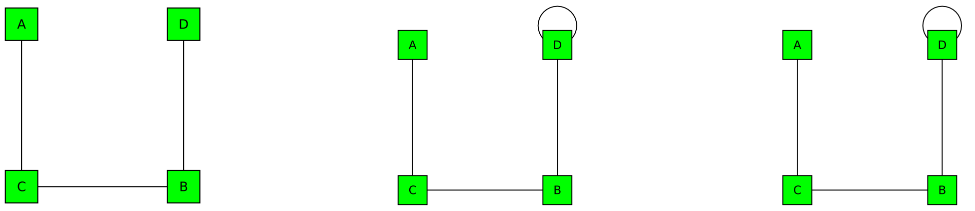
As $\mathcal{V} = S^0 \cup S^1, \mathcal{V} = \{A, B, C, D\}$. So, our graph will have 4 vertices, corresponding to the 4 labels present on the initial labellings. We can now proceed to build the graph iteratively (as the algorithm in section 4.3.2 does it).

The weights of the vertices are initialized as $\mathbf{w}^{(0)} = (f_1, f_2, f_3, f_4) = (18, 19, 19, 0)$.



(a) Step 1 (b) Step 2 (c) Step 3

Figure A.2: Graph construction, for the first three steps.



(a) Step 4 (b) Step 5 (c) Step 6

Figure A.3: Graph construction, for the last three steps.

We now consider the first client. Client 1 is assigned to facility A in $\mathbf{l}^0$ ($l_1^0 = A$) and to facility C in $\mathbf{l}^1$ ($l_1^1 = C$). So, we need to add an edge $\{A, C\}$ to our graph (fig. A.2a). We also need to update our weights. Since $c_{1A} = 10 > 2 = c_{1C}$, facility C has a lower distance to client 1 than facility A. So, we are going to update the weight of facility C by the difference of distances between the two facilities. $w_C^{(1)} = w_C^{(0)} - (c_{1A} - c_{1C}) = 18 - (10 - 2) = 18 - 8 = 10$. Our updated vector of weights is now $\mathbf{w}^{(1)} = (18, 19, 10, 0)$.

Second client: $l_2^0 = A, l_2^1 = C$, but we already have an edge $\{A, C\}$ in the graph (fig. A.2b), so the only thing we need to do is to update the weights. $c_{2A} = 4 < 10 = c_{2C}$. So, $w_A^{(2)} = w_A^{(1)} - (c_{2C} - c_{2A})$. $w_A^{(2)} = 18 - (10 - 4) = 12$. $\mathbf{w}^{(2)} = (12, 19, 10, 0)$.

Third client: $l_3^0 = B, l_3^1 = C$. An edge $\{B, C\}$ is added to the graph (Fig. A.2c). $c_{3B} = 7 < 22 = c_{3C}$. $w_B^{(3)} = w_B^{(2)} - (c_{3B} - c_{3C}) = 19 - (22 - 7) = 4$. $\mathbf{w}^{(3)} = (12, 4, 10, 0)$.

Fourth client: $l_4^0 = D, l_4^1 = B$. An edge $\{D, B\}$ is added to the graph(fig. A.3a). $c_{4D} = 10 < 14 = c_{4B}$. $w_D^{(4)} = w_D^{(3)} - (c_{4D} - c_{4B}) = -4$. Since $w_4^{(4)} < 0$, D must be included in S' (see section 4.3.1 about trivial cases for an explanation). $\mathbf{w}^{(4)} = (12, 4, 10, -4)$.

Fifth client: $l_5^0 = D, l_6^1 = D$. Now, this is a different situation. Since the same label is attributed in two labellings, we create a *self-edge* $\{D, D\}$ (fig. A.3b). As stated in section 4.3.1, this means that the vertex corresponding to D will latter be removed from the graph and D will be included in S'. But D already had to be included in S' since $w_4^{(4)} < 0$. The weights do not need to be updated since $c_{5l_5^0} = c_{5l_5^1}$ (so $\mathbf{w}^{(5)} = \mathbf{w}^{(4)} = (12, 4, 10, -4)$).

The sixth client has a similar situation to fifth. Since $l_6^0 = D, l_6^1 = D$, but an edge $\{D, D\}$ is already in the graph (fig. A.3c), we do not need to update it. We also do not need to update the weights. So, $\mathbf{w} = (12, 4, 10, -4)$.

After having evaluated all clients, we need to remove trivial cases. In this example, we need to remove facility D and its edges. The graph that would be passed to *MIN-WVC* is

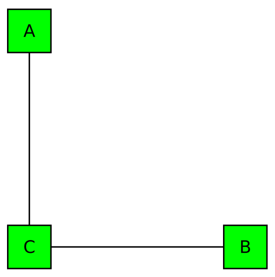


Figure A.4: Graph after removing trivial variables.

in figure A.4.

Appendix B

Example of Transformations on the Fusion Graph

Consider the example in the previous appendix.

$$\mathbf{C} = \begin{bmatrix} 10 & 12 & 2 & 10 \\ 4 & 15 & 10 & 10 \\ 8 & 7 & 22 & 10 \\ 9 & 144 & 15 & 10 \\ 12 & 14 & 8 & 10 \\ 20 & 10 & 22 & 10 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 18 & 19 & 19 & 0 \end{bmatrix}$$

$\mathbf{l}^0 = (A, A, B, D, D, D)$ and $\mathbf{l}^1 = (C, C, C, B, D, D)$.

The weights are exactly the same ($\mathbf{w} = (12, 4, 10, -4)$), only the graph construction differs.

First client has $l_1^0 = A$ and $l_1^1 = C$. So we are going to create an edge (A^0, C^1). Notice how the edge is now oriented. Recalling the biparted minWVC problem given by 4.12, A^0 is the vertex associated with the variable z_A^0.

The second client will not change the graph since it would also introduce the edge (A^0, C^1).

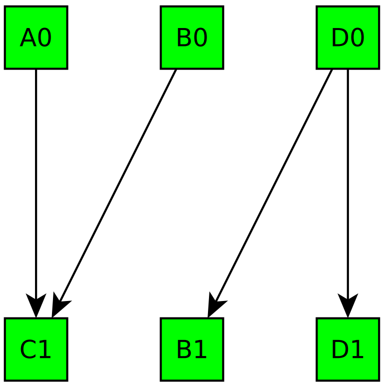
The third client creates the edge (B^0, C^1). Forth client creates the edge (D^0, B^1). Notice how variable B is now duplicated, because it appears in both initial labellings.

The fifth and sixth client introduce the edge (D^0, D^1) (fig. B.1a).

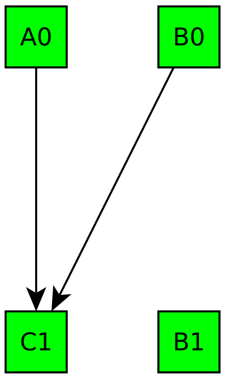
We no longer have self-edges because facilities in different labellings are associated with different vertices. However, we still need to remove vertices associated to facilities with negative weights. In this example, since $w_D = -4 < 0$, it implies removing vertices D^0 and D^1 since both refer to facility D (fig. B.1b).

Vertex B^1 remains in the graph, even though there are no edges connected to it. This is not a problem, since this is a minWVC problem and, therefore, it will never be on a solution. However, facility B is still represented since B^0 is present.

We now proceed to built the network graph that is meant to be solved using the min-



(a) Before.



(b) After.

Figure B.1: $\tilde{G}$ before and after removing trivial variables.

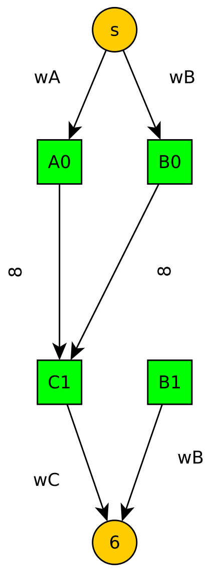


Figure B.2: Graph G_F. The values adjacent to each edge represent their capacity.

cut/max flow optimization. Following the graph construction done by the *MIN-WVC* Algorithm, described in section 4.5, the resulting graph G_F is in figure B.2.

Appendix C

Experimental Dataset for PPR Case Study

Table C.1 bellow describes the pairs of images in the dataset based on the following properties:

- NPlanes GT - Number of planes in the groudtruth solution
- I/ O - (I)ndoor or (O)utdoor sceene.
- Veg - Vegetation is present (Y) or not (N)
- Text Lvl, Texture Level - Texture level of the elements present in the image pair. It can be low, high or variable if the main planes in the scene have different texture levels
- NcP, Non-contiguous Plane - It refers to the existence of two non-contiguous surfaces that are described by the same plane (e.g., a wall with an open gate in the middle).
- LP, Large Planes - refers to the prominence, or not, of the planes present in the scene. Basically, we can have a scene were the major planes occupy a big part of the image or another, were all surfaces are small.

The images pairs that compose the dataset are presented bellow. For each pair, we present the image from the left camera.

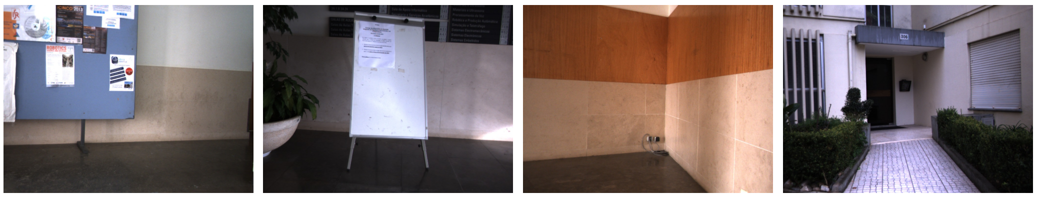

(a) Image 1. (b) Image 2. (c) Image 3. (d) Image 4.

Figure C.1: Images 1-4 of the PPR case study dataset.

Image Pair	NPlanes GT	I/O	Veg	Text Lvl	NcP	LP
1	3	I	N	Variable	N	Y
2	3	I	N	Variable	N	Y
3	4	I	N	Low	N	Y
4	8	O	Y	Variable	N	N
5	3	I	N	Low	N	Y
6	3	I	N	Low	N	Y
7	6	O	N	Low	N	Y
8	6	O	N	Variable	N	N
9	7	O	Y	Variable	N	N
10	3	O	Y	High	N	Y
11	10	O	N	Variable	Y	Y
12	7	O	N	High	Y	Y
13	7	O	Y	Low	Y	Y
14	5	O	N	High	N	Y

Table C.1: Properties of the images in the dataset used for the first case study.

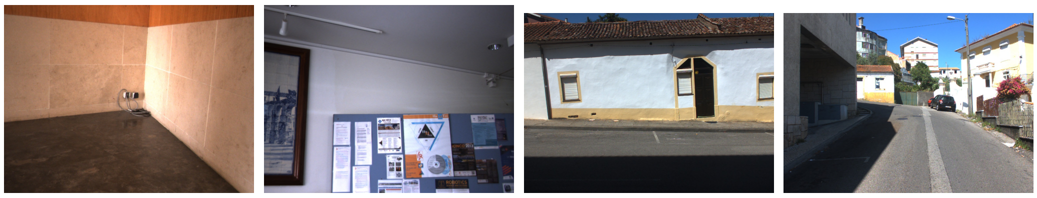

(a) Image 5. (b) Image 6. (c) Image 7. (d) Image 8.

Figure C.2: Images 5-8 of the PPR case study dataset.

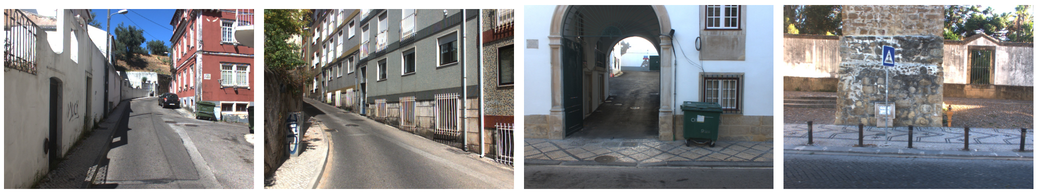

(a) Image 9. (b) Image 10. (c) Image 11. (d) Image 12.

Figure C.3: Images 9-12 of the PPR case study dataset.

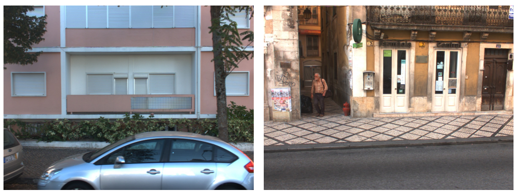

(a) Image 13. (b) Image 14.

Figure C.4: Images 13 and 14 of the PPR case study dataset.

Appendix D

Tables for PPR Experimental Results

The values presented are the average of the parameters (presented in section 5.1.2), for the 50 runs.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	15 ± 0	14485.4 ± 0	42.37 ± 4.7	3.732 ± 0	0.03193 ± 0	20.79 ± 0	37.45 ± 0
F	26.84 ± 2.084	15203.2 ± 129.4	2.065 ± 1.5	4.534 ± 1.5	0.02478 ± 0.0076	16.18 ± 3.1	24.77 ± 5
MPF	14.56 ± 0.7045	14677.8 ± 74.71	7.216 ± 2.7	4.567 ± 1.3	0.02566 ± 0.0072	22.91 ± 4.5	32.2 ± 6.1

Table D.1: Results for image pair 1, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	15 ± 0	13818.3 ± 0	42.5 ± 4.1	3.846 ± 0	0.03344 ± 0	29.34 ± 0	45.82 ± 0
F	23.92 ± 1.688	14396.1 ± 138.5	2.498 ± 1.9	11.67 ± 8.1	0.09377 ± 0.073	20.07 ± 2.6	31.94 ± 7.3
MPF	13.9 ± 0.7071	13985 ± 80.34	6.689 ± 2.1	9.799 ± 7.5	0.1924 ± 0.22	26.62 ± 2.8	39.55 ± 9.3

Table D.2: Results for image pair 2, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	16 ± 0	15307.8 ± 0	52.68 ± 4	2.605 ± 0	0.01858 ± 0	29.61 ± 0	42.3 ± 0
F	25.86 ± 1.471	15909.2 ± 111	4.072 ± 2.2	3.068 ± 3.2	0.01443 ± 0.0073	18.57 ± 3	41.74 ± 3.7
MPF	15.22 ± 0.7365	15467 ± 74.59	8.822 ± 2.9	3.946 ± 5.3	0.01433 ± 0.0067	25.6 ± 4.5	44.39 ± 3.3

Table D.3: Results for image pair 3, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	15 ± 0	11809.8 ± 0	67.68 ± 6.5	17.63 ± 0	0.08561 ± 0	59.52 ± 0	63.42 ± 0
F	25.02 ± 1.61	12417.9 ± 137.4	6.555 ± 2.9	13.77 ± 3.5	0.08396 ± 0.021	40.99 ± 5.1	52.47 ± 6.9
MPF	14.7 ± 0.6776	11948 ± 53.16	11.49 ± 3.5	17.84 ± 5.7	0.1048 ± 0.023	58.97 ± 5.7	66.74 ± 6.9

Table D.4: Results for image pair 4, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	14 ± 0	14875.9 ± 0	44.28 ± 6.9	1.457 ± 0	0.01449 ± 0	28.26 ± 0	67.73 ± 0
F	24.08 ± 1.536	15467 ± 105.2	2.016 ± 1.3	1.996 ± 0.41	0.01216 ± 0.0043	18.43 ± 2.7	55.54 ± 5.7
MPF	14.18 ± 0.5602	15089.1 ± 73.98	6.357 ± 1.7	2.057 ± 0.58	0.01464 ± 0.0044	26.96 ± 2.5	64.22 ± 6

Table D.5: Results for image pair 5, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	15 ± 0	13650.1 ± 0	22.23 ± 0.041	10.43 ± 0	0.006195 ± 0	28.49 ± 0	46.23 ± 0
F	20.56 ± 1.692	13980.1 ± 68.06	0.454 ± 0.1	10.26 ± 2.2	0.01562 ± 0.0054	23.49 ± 3.5	36.58 ± 8.3
MPF	13.86 ± 0.5349	13785.7 ± 47.86	3.718 ± 0.4	10.63 ± 2.2	0.01478 ± 0.006	28.05 ± 4.6	39.33 ± 9.6

Table D.6: Results for image pair 6, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	16 ± 0	14679.5 ± 0	65.79 ± 0.49	7.881 ± 0	0.06615 ± 0	40.7 ± 0	48.14 ± 0
F	28.64 ± 1.675	15393.2 ± 164.4	6.893 ± 2.8	5.61 ± 1.4	0.07699 ± 0.0091	29.71 ± 4	43.17 ± 5.1
MPF	14.8 ± 0.6999	14809.2 ± 123.7	12.17 ± 3	6.594 ± 0.8	0.07252 ± 0.008	43.78 ± 4	53.83 ± 4.4

Table D.7: Results for image pair 7, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	19 ± 0	15285.1 ± 0	104.7 ± 31	21.56 ± 0	0.4413 ± 0	36.15 ± 0	55.07 ± 0
F	31.82 ± 2.047	16000.3 ± 197.2	12.35 ± 7.3	24.39 ± 4.3	0.3965 ± 0.009	26.23 ± 2.3	48.72 ± 3.2
MPF	18.2 ± 0.8806	15408.3 ± 69.03	20.34 ± 8.9	25.07 ± 5.4	0.4188 ± 0.018	36.84 ± 3.3	55.37 ± 2.7

Table D.8: Results for image pair 8, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	17 ± 0	14341.2 ± 0	125.5 ± 34	13.04 ± 0	0.1335 ± 0	44.68 ± 0	65.56 ± 0
F	28.18 ± 1.6	14920.9 ± 173.7	16.71 ± 7.5	13.19 ± 1	0.1456 ± 0.0097	29.93 ± 2.5	62.17 ± 3.3
MPF	17.7 ± 0.7354	14470.4 ± 58.06	23.88 ± 7.5	13.22 ± 0.7	0.1424 ± 0.0096	42.11 ± 2.7	68.86 ± 3.6

Table D.9: Results for image pair 9, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	10 ± 0	12375.8 ± 0	117.8 ± 39	0.8018 ± 0	0.007776 ± 0	32.09 ± 0	81.59 ± 0
F	18.22 ± 1.375	12813 ± 155	10.46 ± 7.7	0.8551 ± 0.32	0.009856 ± 0.0019	23.65 ± 3.5	81.93 ± 7
MPF	10.36 ± 0.6627	12433.5 ± 73.92	14.75 ± 8.8	0.8141 ± 0.27	0.01002 ± 0.0022	31.88 ± 3.3	91.6 ± 7.6

Table D.10: Results for image pair 10, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	18 ± 0	14108.7 ± 0	155.2 ± 53	3.735 ± 0	0.03792 ± 0	45.6 ± 0	67.39 ± 0
F	30.26 ± 1.747	14914 ± 174.5	15.85 ± 8.8	6.627 ± 2.4	0.07238 ± 0.02	37.99 ± 4	57.85 ± 5.9
MPF	16.4 ± 0.833	14307.1 ± 107.7	22.12 ± 12	6.418 ± 3.1	0.07736 ± 0.023	52.39 ± 6.4	69.15 ± 6.6

Table D.11: Results for image pair 11, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	18 ± 0	13869.8 ± 0	171 ± 55	4.037 ± 0	0.0885 ± 0	31.62 ± 0	44.72 ± 0
F	32 ± 2.321	14685.8 ± 232.8	24.4 ± 11	2.591 ± 1.3	0.08082 ± 0.0051	28.57 ± 3.1	44.02 ± 4.9
MPF	16.78 ± 0.91	13990.3 ± 94.61	35.15 ± 17	2.203 ± 0.88	0.08325 ± 0.0043	42.17 ± 6.2	56.03 ± 6.3

Table D.12: Results for image pair 12, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	21 ± 0	17057.7 ± 0	204.7 ± 17	7.095 ± 0	0.02192 ± 0	29.07 ± 0	34.16 ± 0
F	36.84 ± 2.226	17978.3 ± 174.9	15.6 ± 10	8.414 ± 1.4	0.0271 ± 0.0085	23.23 ± 2.6	25.63 ± 3.5
MPF	19.86 ± 0.926	17245.1 ± 115.8	24.11 ± 11	8.691 ± 1.7	0.02653 ± 0.0091	32.65 ± 3	35.63 ± 4.4

Table D.13: Results for image pair 13, with the 3 algorithms.

Algo	NPlanes	Energy	T Exec	Ang. Dev.	Ave. h	Attrb. (%)	Attrb. R. (%)
MP	12 ± 0	12337.2 ± 0	122.9 ± 8.9	0.7887 ± 0	0.01058 ± 0	43.94 ± 0	88.1 ± 0
F	21.74 ± 2.211	13006.9 ± 215.3	14.34 ± 6.6	1.371 ± 2.4	0.01703 ± 0.014	36.52 ± 5.7	61.95 ± 5.4
MPF	10.96 ± 0.6047	12516.5 ± 89.89	16.76 ± 6.3	1.983 ± 4.3	0.01633 ± 0.0068	53.8 ± 2.9	72.52 ± 8.5

Table D.14: Results for image pair 14, with the 3 algorithms.

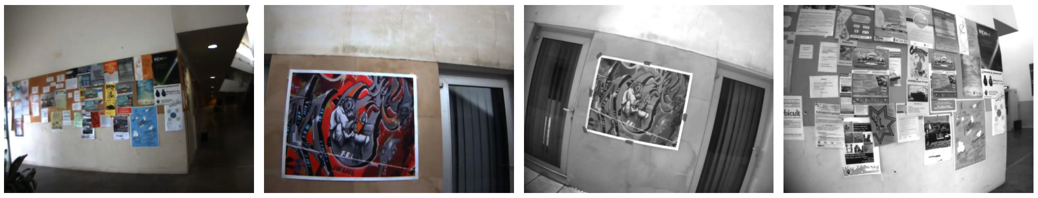
Appendix E

Experimental Dataset for the UIC Case Study

Table E.1 bellow presents the images used in the second case study. The image properties presented in table are the following:

- Image - Image Index
- NSeg - Number of detected segments in the image
- I/O - Image presents an (I)ndoor or (O)utdoor scene
- Co-linear segs - (Y) if co-linear segments are present in the image (refer to section 5.2.2 for further explanations) and (N) otherwise
- Camera - Camera used for the acquisition of the images

The images used are presented bellow.



(a) Image 1. (b) Image 2. (c) Image 3. (d) Image 4.

Figure E.1: Images 1-4 of the UIC case study dataset.

(a) Image 5.

(b) Image 6.

(c) Image 7.

(d) Image 8.

Figure E.2: Images 5-8 of the UIC case study dataset.

(a) Image 9.

(b) Image 10.

(c) Image 11.

(d) Image 12.

Figure E.3: Images 9-12 of the UIC case study dataset.

(a) Image 13.

(b) Image 14.

(c) Image 15.

(d) Image 16.

Figure E.4: Images 13-16 of the UIC case study dataset.

Figure E.5: Image 17 of the UIC case study dataset.

Image	NSeg	I/O	Co-linear segs	Camera
1	46	I	N	KLT10
2	87	O	N	KLT10
3	61	O	Y	KLT25
4	126	I	N	KLT25
5	35	I	Y	KLT25
6	48	I	N	KLT45
7	84	O	N	KLT45
8	170	I	N	FishEye
9	192	I	N	FishEye
10	294	I	N	FishEye
11	78	I	Y	GoPro
12	317	O	N	GoPro
13	414	O	N	GoPro
14	777	O	N	GoPro
15	534	O	N	GoPro
16	85	I	Y	GoPro
17	554	I	Y	GoPro

Table E.1: Properties of the images in dataset used for the second case study.

Appendix F

Tables for the UIC experimental results

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
1	MP	0.145 ± 0	0.00289 ± 0	0.019 ± 0	0.101 ± 0.018	19.6 ± 0	90
	F	0.143 ± 0.0049	0.00889 ± 0.019	0.0182 ± 0.0025	0.00653 ± 0.0009	19.7 ± 0.15	

Table F.1: Results for image pair 1, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
2	MP	0.194 ± 0	0.0939 ± 0	0.0595 ± 0	0.403 ± 0.082	72.4 ± 0	100
	F	0.194 ± 0	0.0939 ± 0	0.0595 ± 0	0.0183 ± 0.0043	72.4 ± 0	

Table F.2: Results for image pair 2, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
3	MP	0.0137 ± 0	0.0393 ± 0	0.0779 ± 0	1.29 ± 0.079	106 ± 0	20
	F	0.0297 ± 0.016	0.02 ± 0.018	0.0531 ± 0.024	0.0187 ± 0.0037	108 ± 1.6	

Table F.3: Results for image pair 3, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
4	MP	0.612 ± 0	0.0498 ± 0	0.364 ± 0	0.294 ± 0.1	39.9 ± 0	100
	F	0.612 ± 0	0.0498 ± 0	0.364 ± 0	0.0235 ± 0.016	39.9 ± 0	

Table F.4: Results for image pair 4, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
5	MP	1.1 ± 0	0.561 ± 0	0.426 ± 0	0.472 ± 0.052	63.9 ± 0	100
	F	1.1 ± 0	0.561 ± 0	0.426 ± 0	0.0182 ± 0.0019	63.9 ± 0	

Table F.5: Results for image pair 5, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
6	MP	0.26 ± 0	0.0661 ± 0	0.103 ± 0	0.118 ± 0.025	27.2 ± 0	100
	F	0.26 ± 0	0.0661 ± 0	0.103 ± 0	0.00634 ± 0.00094	27.2 ± 0	

Table F.6: Results for image pair 6, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
7	MP	0.0608 ± 0	0.0383 ± 0	0.00741 ± 0	0.345 ± 0.054	64.9 ± 0	100
	F	0.0608 ± 0	0.0383 ± 0	0.00741 ± 0	0.0182 ± 0.0029	64.9 ± 0	

Table F.7: Results for image pair 7, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
8	MP	0.146 ± 0	0.042 ± 0	0.0468 ± 0	1.49 ± 0.13	133 ± 0	100
	F	0.146 ± 0	0.042 ± 0	0.0468 ± 0	0.0599 ± 0.0076	133 ± 0	

Table F.8: Results for image pair 8, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
9	MP	0.315 ± 0	0.0579 ± 0	0.0678 ± 0	1.15 ± 0.13	120 ± 0	80
	F	0.416 ± 0.21	0.0484 ± 0.02	0.0796 ± 0.025	0.0415 ± 0.0072	120 ± 0.12	

Table F.9: Results for image pair 9, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
10	MP	0.51 ± 0	0.171 ± 0	0.0289 ± 0	0.49 ± 0.0038	91.2 ± 0	100
	F	0.51 ± 0	0.171 ± 0	0.0289 ± 0	0.0252 ± 0.0006	91.2 ± 0	

Table F.10: Results for image pair 10, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
11	MP	0.0161 ± 0	0.046 ± 0	0.074 ± 0	2.17 ± 0.23	119 ± 0	40
	F	0.0523 ± 0.074	0.0232 ± 0.024	0.0746 ± 0.05	0.0319 ± 0.0045	121 ± 1.3	

Table F.11: Results for image pair 11, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
12	MP	0.42 ± 0	0.109 ± 0	0.182 ± 0	12.4 ± 0.45	444 ± 0	50
	F	0.338 ± 0.097	0.121 ± 0.013	0.221 ± 0.041	0.19 ± 0.027	446 ± 2.6	

Table F.12: Results for image pair 12, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
13	MP	0.0325 ± 0	0.0733 ± 0	0.00145 ± 0	9.09 ± 0.65	324 ± 0	20
	F	0.0569 ± 0.013	0.16 ± 0.046	0.0205 ± 0.01	0.159 ± 0.033	326 ± 0.69	

Table F.13: Results for image pair 13, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
14	MP	0.0597 ± 0	0.0446 ± 0	0.0484 ± 0	4.71 ± 0.081	225 ± 0	30
	F	0.115 ± 0.091	0.0708 ± 0.06	0.121 ± 0.13	0.0957 ± 0.016	228 ± 2.8	

Table F.14: Results for image pair 14, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
15	MP	0.0271 ± 0	0.0164 ± 0	0.00863 ± 0	2.78 ± 0.094	223 ± 0	50
	F	0.074 ± 0.056	0.053 ± 0.075	0.031 ± 0.029	0.0329 ± 0.003	225 ± 1.6	

Table F.15: Results for image pair 15, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
16	MP	0.0552 ± 0	0.118 ± 0	0.0895 ± 0	2.77 ± 0.41	200 ± 0	10
	F	0.034 ± 0.019	0.0306 ± 0.031	0.0557 ± 0.025	0.0355 ± 0.0051	204 ± 2	

Table F.16: Results for image pair 16, with the 2 algorithms.

Im Id	Algo	Dist2GT			TExec	Energy	Convergence(%)
		eta	cx	cy			
17	MP	0.0279 ± 0	0.00991 ± 0	0.034 ± 0	3.99 ± 0.17	187 ± 0	80
	F	0.0292 ± 0.0029	0.00885 ± 0.0022	0.0328 ± 0.0025	0.0619 ± 0.013	191 ± 1.5	

Table F.17: Results for image pair 17, with the 2 algorithms.

Bibliography

[1] N. Lazic, I. E. Givoni, B. J. Frey, and P. Aarabi, "Floss: Facility location for subspace segmentation," *ICCV'09*, pp. 825–832, 2009.

[2] C. Raposo, M. Antunes, and J. Barreto, "Piecewise-planar stereoscan: Structure and motion from plane primitives," *Computer Vision – ECCV 2014*, vol. 8690, pp. 48–63, 2014.

[3] R. Melo, M. Antunes, J. Barreto, G. Falcao, and N. Goncalves, "Unsupervised intrinsic calibration from a single frame using a "plumb-line" approach," *Computer Vision (ICCV), 2013 IEEE International Conference on*, 2013.

[4] A. Delong, O. Veksler, and Y. Boykov, "Fast fusion moves for multi-model estimation," *Proceedings of the European Conference on Computer Vision*, 2012.

[5] S. Sinha, "Graph Cut Algorithms in Vision, Graphics and Machine Learning An Integrative Paper," *UNC Chapel Hill*, 2004.

[6] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, pp. 1124–37, Sept. 2004.

[7] N. Lazic, B. J. Frey, and P. Aarabi, "Solving the uncapacitated facility location problem using message passing algorithms," *Journal of Machine Learning Research - Workshop and Conference Proceedings*, vol. 9, pp. 429–436, 2010.

[8] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, pp. 147–59, Feb. 2004.

[9] H. Isack and Y. Boykov, "Energy-based geometric multi-model fitting," *International Journal of Computer Vision*, vol. 97, no. 2, pp. 123–147, 2012.

[10] T. Cormen, C. Leiseron, R. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press.

[11] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, 1998.

[12] D. Greig, B. Porteous, and A. Seheult, “Exact maximum a posteriori estimation for binary images,” *Journal of the Royal Statistical Society*, vol. 51, no. 2, pp. 271–279, 1989.

[13] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.

[14] K. Spielberg, “Algorithms for the simple plant-location problem with some side conditions,” *Operations Res. 17*, 1969.

[15] J. Krarup and P. Pruzan, “The simple plant location problem: survey and synthesis,” *European Journal of Operational Research*, 1983.

[16] M. Balinsky and P. Wolfe, “On Benders decomposition and a plant location problem, Working paper ARO-27,” 1963.

[17] M. Balinsky, “On finding integer solutions to linear programs,” *Proc. IBM Scientific Symposium on Combinational Problems*, 1966.

[18] A. Kuhen and H. M.J., “A heuristic program for locating warehouses,” *Management Sci.*, 1963.

[19] S. Reiter and G. Sherman, “Allocating indivisible resources affording external economies or diseconomies,” *Internat. Econom. Rev. 3*, 1962.

[20] M. Guigndard and S. K., “Algorithms for exploiting the structure of the simple plant location problem,” *Ann. Discrete Math. I*, 1977.

[21] A. Weber and C. Friedrich, *Theory of the location of industries*. Materials for the study of business, University of Chicago Press, 1929.

[22] I. Dinur and S. Safra, “The importance of being biased,” *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pp. 33–42, 2002.

[23] M. Antunes and J. Barreto, “Semi-dense piecewise planar stereo reconstruction using SymStereo and PEARL,” *3D Imaging, Modeling, Processing, . . .* , pp. 230–237, Oct. 2012.

[24] M. Antunes, *Stereo Reconstruction using Induced Symmetry and 3D scene priors*. PhD thesis, University of Coimbra, 2014.

[25] J. Barreto, J. Roquette, P. Sturm, and F. Fonseca, “Automatic Camera Calibration Applied to Medical Endoscopy,” in *BMVC 2009 - 20th British Machine Vision Conference* (A. Cavallaro, S. Prince, and D. C. Alexander, eds.), (London, United Kingdom), pp. 1–10, The British Machine Vision Association (BMVA), Sept. 2009.