Divide & Clean

A Master's Thesis

L.F.O.Vogels

Multi-constrained edge partitioning and its application in debris management

TU Delft

Divide & Clean

A Master's Thesis

by

L.F.O.Vogels

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday, December 19, 2019 at 15:00.

Student number:	4691547	
Project duration:	March 1, 2019 – December 19, 2019	
Thesis committee:	Dr. P. Keskinocak	Georgia Tech, supervisor
	Prof. dr. ir. K. Aardal,	TU Delft, supervisor
	Dr. ir. M. van Gijzen,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Georgia Institute of Technology

Preface

I would like to thank three people who made my master's possible. First, Dr. Van Gijzen. I remember walking into your office in 2016 to ask whether I could do my master's at the TU Delft. It is great that you are here when I finish it. Thank you for your humour and for being approachable. Then, Prof. Aardal. You recommended me to apply for Georgia Tech and guided me every step of the way. Outside my thesis, you also made time to talk about my future. Thank you for your time and devotion. Lastly, Dr. Keskinocak. You have been incredible. You barely knew me, but decided anyway to make time for me every week for nine months. Your input was indispensable for this thesis. You made me feel at home in Atlanta and you were flexible and compassionate when I had to return home earlier than expected. Thank you for everything.

L.F.O.Vogels
Delft, December 2019

Abstract

Let $G = (V, E)$ be a connected undirected graph, where every edge has two weights assigned to it. This thesis considers the partitioning of the edge set E of G into subsets with three objectives in mind: i) balance the total amount of the first weight among the subsets, ii) balance the total amount of the second weight among the subsets and iii) create a non-chaotic partition. Here non-chaotic means that every subset forms a distinguishable and compact subgraph. We call this Unrelated Unconnected Multi Constrained Graph Partitioning, or UUMCGP. It has applications in the collection of debris after natural disasters and in the assignment of tasks to computers in a distributed network. We are the first to define UUMCGP and we show that for specific cases close-to-optimal solutions can be obtained in polynomial time. Moreover, an algorithm is developed that gives good approximate solutions to the general case of UUMCGP. The algorithm is tested on real-life cases of debris management and gives better results than commercial solvers when they are set to find the optimal solution.

Contents

1 Introduction

In 2005 Hurricane Katrina buried New Orleans in an amount of debris equivalent to the area of a football pitch 10 kilometers high [24]. Clearance operations were complicated by the sheer mass of the destruction and diversity of debris including trees, electronic appliances and other hazardous waste. It took over 10 years to return New Orleans to its previous state. After natural disasters, whether they be earthquakes, hurricanes or floods, it has been proven time and time again that debris collection is a long and arduous process, accounting for 27% of the total cost of debris management [34]. Combined with the fact that now more people are affected by natural disasters than ever [28], leads us to conclude that efficient debris collection is extremely important.

Debris clearance comprises two phases. First, the debris is moved to the side of the road to allow emergency services to operate. In the second phase, the debris is transferred to processing facilities by independent contractors. The local government assigns each street to a particular contractor. An assignment results in (i) an operation time for each contractor, (ii) a profit for each contractor and (iii) a "chaos metric", which we now loosely define as the clarity of the assignment when glanced upon by an observer.

A problem in a totally different category than the debris management problem, but with a strong mathematical resemblance to it, is the processor problem. Here nodes of a graph represent tasks to be executed by computers located on other fixed nodes in the graph. Each task requires memory and computing time to execute. Moreover, the assignment of tasks to computers needs to be done in such a way that the communication costs between the computer and the tasks assigned to it are minimized. This leads to the desire to have a partitioning of the graph in such a way that each computer has its own connected area of tasks assigned to it. Note that the processor problem is in fact a node partitioning problem. However, by dividing the load of a node over its adjacent edges, we can reformulate the problem to an edge-partitioning problem equal to the debris management problem.

In this thesis we abstract the issues of the debris management and processor problems to a mathematical program that we will call Unrelated Unconnected Multi Constrained Graph Partitioning (UUMCGP). Here, we assign the edges of a graph to agents. When an edge gets assigned to an agent, that agent will perform the tasks related to that edge. It will take an agent two types of resources to execute that task. Examples of two resources could be time and money in the case of debris management. In the case of assigning jobs to computers, memory and computing time are examples of the two resources. Each agent will use these resources at a different rate. In the debris management problem some contractors will clean a street faster and cheaper for example. The same goes for computers. Some of them might be faster or have more memory available. When all edges are assigned, each agent will have two total loads. The first (second) load corresponds to the sum, over all edges, of the first (second) resource that agent uses to process all edges assigned to him. Each assignment of edges to agents can also be seen as a partition of the graph. A partition with certain "niceness" properties. An example of a niceness property is the number of nodes that have adjacent edges assigned to different agents. Another example could be the number of zones, where a zone is a subset of the edges assigned to the same agent.

In UUMCGP we assign edges to agents with three objectives in mind. First, the partition should balance the total loads of the *first* resource of all agents. We measure the fit of this balance by the maximum total load of the first resource for any agent. We try to minimize this maximum load. In the debris management problem this would mean that we are trying to minimize the time that all contractors are finished cleaning the streets assigned to them. In the processor problem this objective resembles the wish to minimize the time for all computers to complete their tasks.
Second, the partition should balance the total loads of the *second* resource of all agents. We measure the fit of this balance by the minimum total load of the second resource for any agent. We try to maximize this minimum load. In the debris management problem this would mean that we are trying to maximize the profit of the least earning contractor.
Third, the partition should be "nice" or non-chaotic. This means roughly that the graph with all edges assigned gives an organized impression to the observer. In the debris management problem it resembles the wish that contractors have clear areas of the city assigned to them. It is certainly unwanted for example to have one neighbourhood assigned to ten different contractors. In the aftermath of a disaster it would add to the chaos if trucks of ten different contractors drive around in the same area. In the processor problem this objective reflects the desire to minimize the communication costs. To put this loosely defined desire in a more robust mathematical framework, we will define metrics for the chaos of an assignment later on. The third objective then comes down to minimizing one of these metrics.

This thesis will answer the following questions regarding UUMCGP:

1. Can we define UUMCGP as an MIP?
2. Can we solve this MIP in polynomial time?
3. Can we come up with good approximate solutions to this MIP using an algorithm?
4. How does this algorithm perform in real-life cases of debris management?

In the next chapter we will summarize the literature of problems similar to UUMCGP. We will then introduce notation and write out the MIP of UUMCGP in Chapter 3. In Chapter 4 we will see that UUMCGP is solvable in polynomial time in some specific cases. An algorithm for UUMCGP is designed in Chapter 5, which is tested on 9 disaster prone cities in the United States using the experimental design described in Chapter 6. The results of these experiments are shown in Chapter 7. Finally, we conclude our findings in Chapter 8 and give suggestions for further research in Chapter 9.

2

Literature review

As discussed in the introduction, this thesis considers the problem of Unrelated Unconnected Multi Constrained Graph Partitioning (UUMCGP). Here, we assign edges of a graph to agents with three objectives:

1. Balance the total loads of the first resource among all agents.
2. Balance the total loads of the second resource among all agents.
3. Minimize the chaos.

In this chapter we will see that literature is abundantly available when only one or two of these objectives are considered. Despite its practical applications, to the best of our knowledge and with the exception of one paper [31], there has not been any research including all three objectives.

When only the first objective is considered, the problem falls under the category of parallel machine scheduling. These problems consider assigning jobs to machines with the objective to minimize the total operation time, also called the makespan. Each job has a different processing time for each machine. Each job must be assigned to exactly one machine. Each job is ready to be processed right from the start and there are no constraints regarding the finish time of a job. Neither are there any precedence constraints for the jobs. An example of this problem is given in Figure 2.1.

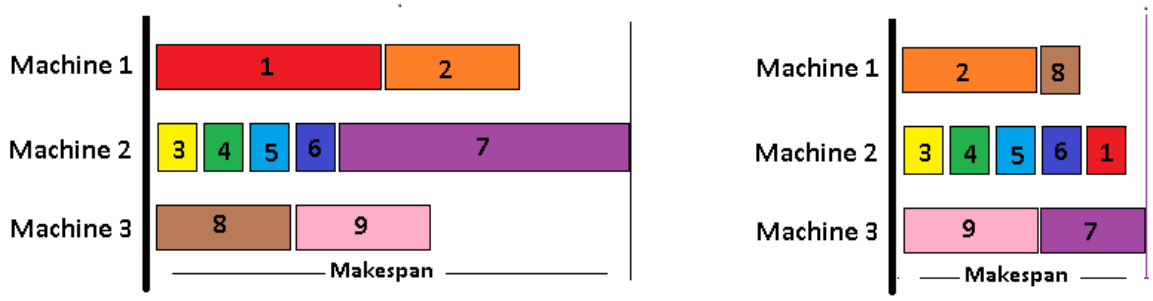


Figure 2.1: Two solutions to a minimum makespan problem $R_m||C_{max}$ with 9 jobs and 3 machines. Jobs have different processing times for different machines. Therefore, the different arrangement in the second solution has a lower makespan.

Parallel machine scheduling, using the notation of [13], can be denoted by $R_m||C_{max}$ and is NP-hard, since the special case with identical machines is known to be so [9]. Since its introduction by McNaughton in 1959 [25], methods have been proposed to solve the problem. In [30] a branch and bound method is given to solve the problem to optimality. However, this method has practical limitations on larger instances. Due to the NP-hardness of the problem, many approximation algorithms have been proposed to solve the problem approximately, starting with Ibarra and Kim, who propose six m-approximation algorithms, where m is the number of machines [18]. The best approximation algorithm so far is that of Lenstra et al. [22]. They give a 2-approximation algorithm and showed that no polynomial time algorithm exists with a better worst-case ratio than 3/2 unless $P = NP$. It is this 2-approximation algorithm that serves as a basis for one of the theorems in this paper and for the development of an algorithm for UUMCGP.

We can also include both the first and the second objective. We can do this in two ways.
The first way entails including the second objective as a constraint in our assignment problem $R_m||C_{max}$. Here, we still minimize the makespan, but we are constrained by a resource whose availability is limited. Every execution of a job on a machine requires using that resource. This problem is similar to unrelated parallel machine scheduling with additional resource constraints, or RCPMSP. There is significantly less research done with this extra resource constraint added in the scheduling problems. Edis [5] summarizes the complexity and solution methods of this field. It should be noted that in the RCPMSP, resource constraints need to be met at any moment in time. In our case, however, we are only interested in the total resource use of a job machine assignment.
The second way involves including both the first and second objective as constraints in our problem. We then add costs for the execution of every job by an agent and minimize the sum of all costs. The problem we end up with is the multi-resource generalized assignment problem, or MRGAP [11]. An example of a feasible solution to MRGAP is given in Figure 2.2.

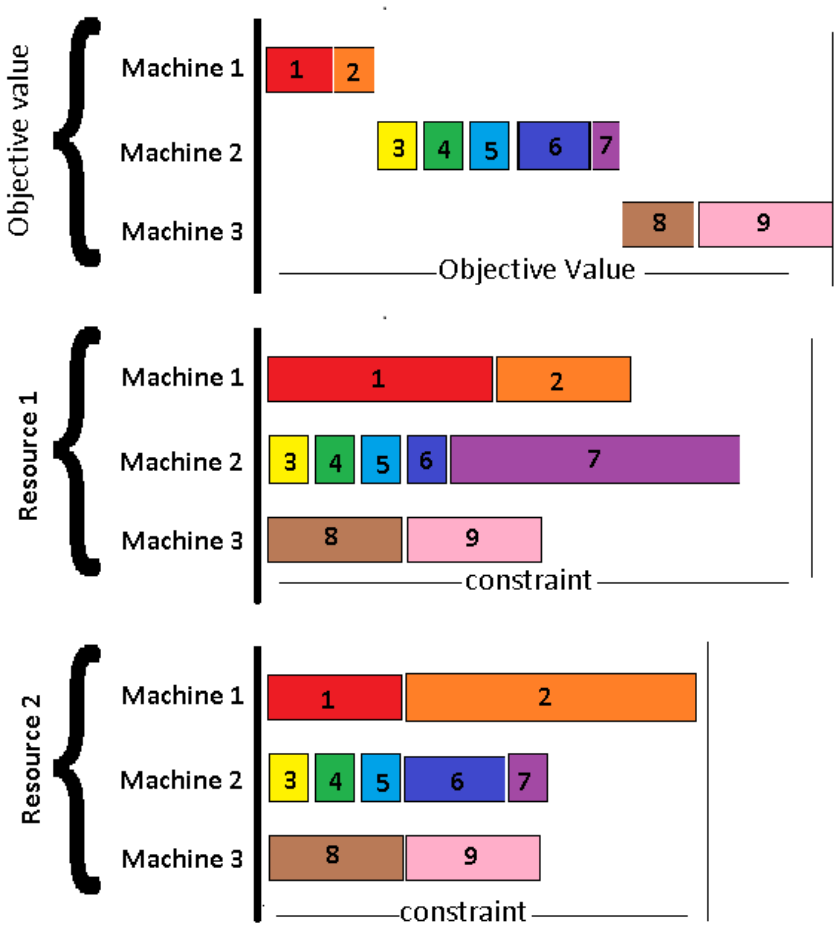


Figure 2.2: A feasible solution to MRGAP with three machines, nine jobs, and two resources. This solution is feasible, since for every resource, every machine uses less than the amount available. The objective value needs to be minimized.

MRGAP is an extension of the generalized assignment problem, GAP. GAP can be seen as Figure 2.2 with only one resource. MRGAP is known to be NP-hard [29]. The simpler problem GAP is NP-complete, since the partition problem can be reduced to GAP on two machines. For MRGAP, Gavish and Pirkul proposed a branch-and-bound algorithm and two simple Lagrangian heuristics [11]. The only other algorithm that is developed is by Yagiura et al. [36]. They use tabu search and a very large-scale neighborhood search to come up with an effective algorithm. In the structural result chapter of this thesis, we will show that by conditioning

parameters and allowing the constraints to be relaxed mildly, the MRGAP becomes polynomially solvable. This result is used when designing our algorithm.

So far, we have not considered the third objective. When we do include this objective, we enter the extensive field of graph partitioning. We will now categorize the available literature in this field to position our problem. We will categorize each relevant class of partition problems according to five characteristics of graph partitioning: the objects of partition, connectivity of partition, the number of balancing constraints, the type of the weights to be balanced, and the type of niceness to be optimized. We will first elaborate on each of these characteristics. Then, the relevant classes will be positioned using their values for these characteristics, resulting in Table 2.1.

2.1. Characteristics

Objects of partition
The first characteristic of a graph partitioning problem is the objects of partition. There are two possible objects of partition: nodes and edges. In node partitioning the goal is to split the node set V of a graph in γ disjoint subsets $V_1, V_2, \ldots, V_\gamma$ for some γ. Similarly, in edge partitioning, the edge set E of a graph is split in γ disjoint subsets $E_1, E_2, \ldots, E_\gamma$ for some γ.

Connectivity
The second characteristics is the connectivity of the partition assigned to one agent. We will see that in most graph partitioning literature, for all agents k, the subset of nodes or edges assigned to k is required to be connected.

Number of balancing constraints
Thirdly, we will categorize graph partitioning problems by the number of balancing constraints. A balancing constraint requires the subsets to be of similar size. Most partitioning problems have one such balancing constraint, but there are problems that include several balancing constraints.

Weight type
Fourthly, the type of weights in the balancing constraints will be looked at. We mentioned that a balancing constraint requires the subsets to be of similar size. Suppose the subset $V' \subset V$ is assigned to agent k. The size $S(V')$ of V' is now defined by $S(V') = \sum_{v \in V'} w_{v,k}$, for given $w_{v,k} \in \mathbb{R}$. In line with [13], we consider three weight types:

- Unit: $w_{v,k} = 1$ for all v, k, in this case the size of a subset is simply the number of elements in the subset.
- Identical: $w_{v,k} = t_v$ for all k, here every element v has a different weight t_v. However, the weights of an element are the same for every agent.
- Unrelated: there is no rule for the weight value $w_{v,k}$.

Niceness type
Lastly, we consider the niceness objective to be optimized. In all partitioning problems the objective to be optimized has to do with the niceness of the solution. This niceness is defined in different ways throughout the literature. We consider three of these ways.

- Edge cut: niceness is the sum of the weights of the edges with nodes in different partitions.
- Node span: niceness is the sum of the span of all the nodes, where the span of a node is the number of different agents assigned to edges adjacent to that node. In Figure 2.3 an example shows the span of some nodes. In Expression (3.2) in the next chapter, we will formally define the span of a node.
- Shape: the geometric shape of the subsets.

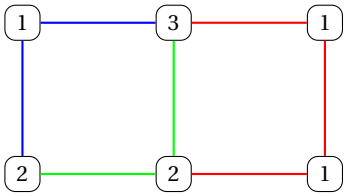


Figure 2.3: An example graph showing the span of each node.

2.2. Classes

k-cut problem
The most simple of the classes of graph partitioning is the k-cut problem. The problem is to divide the nodes of a graph in k connected subsets such that the sum of the weights of the edges that have nodes in different subsets is minimized. There is no constraint on the size of the subsets. The problem can be solved in polynomial time for fixed k, [9].

Classical graph partitioning
Classical graph partitioning is similar to the k-cut problem, but with one balancing constraint added. It has no polynomial time solution. The first effective algorithm designed for this problem was proposed by Kernighan and Lin [21].

Political Districting
A similar problem is studied in the field of political districting. Here, a state or country has to be divided into parts of equal population size, where each part should have a so-called compact shape. Here, compactness means that the shape should be circular or square-like instead of long, thin or awkwardly shaped. Several solution methods have been proposed for political districting problems including exact [10], heuristic [26], and metaheuristic [4] methods.

Multi-Constrained Graph Partitioning
The first class of graph partitioning problems that has multiple balance constraints is Multi-Constrained Graph Partitioning, or MCGP. Here, each edge has n weights, each corresponding to a resource. The nodes now need to be partitioned such that the subsets are balanced for all n resources, while minimizing the edge cut. The first heuristics of its kind were developed in 1998 by Karypis and Kumar [20].

Edge partitioning
So far, we have only considered node partitioning. The next class does something different. It partitions the edges. It is a relatively novel field and rose in importance when it was observed that it is a good approach to solve the processor problem discussed in Chapter 1, for a large number of computers. The goal is to partition the set of edges into connected subsets such that the total edge weight for each subset is balanced, while minimizing the sum of the span of all nodes. Results on edge partitioning can be found in [12], [17], [15]. In [3] the relation between node and edge partitions is shown and approximation algorithms for the problem are given.

UUMCGP
In this paper, we consider the self-named Unconnected Unrelated Multi-Constrained Graph Partitioning, or UUMCGP. The main difference with this problem and existing literature is two-fold. First, the edge weights are unrelated. This means that every edge has a different weight for each agent. This reflects the fact that in the debris management problem some agents are faster with cleaning the debris of streets than others. In the computer processing problem it reflects that some processors are more efficient or require less electricity than others. Secondly, there is no connectivity of the subsets required. In the debris management problem, for example, an agent can be assigned unconnected neighbourhoods of the city. To the best of our knowledge [31] is the only research done similar to UUMCGP. However, it differs in three ways from this research. First, their focus is on the application to debris management, while we take a general approach. Second, the author's solution method involves a human guided tool while we focus only on computerized solution methods. Third, they consider facilities, whereas in this thesis, facilities are not taken into account. Therefore they are dealing with a double assignment problem, where they assign each street to both an agent ánd a facility. Moreover, in their case, the operation time also depends on the facility processing times.

Class	Papers	Object of Partition	Connectivity required	#Balancing constraints	Weight type	Niceness type
k-cut	[9]	Nodes	yes	0	NA	Edge cut
Classical Graph Partitioning	[21]	Nodes	yes	1	Identical	Edge cut
Political Districting	[10], [26], and [4]	Nodes	yes	1	Identical	Shape
MCGP	[20]	Nodes	yes	n	Identical	Edge cut
Edge partitioning	[12], [17], [15] and [3]	Edges	yes	1	Identical	Span
UUMCGP	[31],this thesis	Edges	no	2	Unrelated	Span

Table 2.1: Overview of relevant graph partitioning classes with characteristics.

2.3. Contribution

Our contribution will be three-fold. First, we state UUMCGP, prove structural results for it and design an algorithm. We are the first to do so. Secondly, we expand the field of multi-resource generalized assignment problems (MRGAP) by designing an algorithm that solves MRGAP while only slightly violating the constraints. Lastly, the algorithm we develop for UUMCGP is, when applied to the debris management problem, the first algorithm to be successfully applied to nine different cities in the United States.

3

Problem Description and Notation

In Chapter 1 we introduced two problems that motivated this paper: the debris management problem and the processor problem. Both problems are real-life examples of Unconnected Unrelated Multi-Constraint Graph Partitioning. In this chapter we define UUMCGP as a mixed-integer program, or MIP. We will denote this program by $P_1(G)$. To formulate this program, we use the notation in Table 3.1.

Decision variables:

$$x_{e,k} = \begin{cases} 1 & \text{if edge } e \text{ gets assigned to agent } k \\ 0 & \text{otherwise} \end{cases}$$

Sets:

V	Set of nodes $v \in \{1,2,..,n\}$
E	Set of edges $e \in \{1,2,..,m\}$
K	Set of agents $k \in \{1,2,..,r\}$

Parameters:

$G = (V,E)$	Undirected connected graph
$w^{(1)}_{e,k}$	First edge weight of edge e for agent k
$w^{(2)}_{e,k}$	Second edge weight of edge e for agent k
$g(G,x)$	function mapping the combination of a graph G and a partition x to a so-called chaos number

Objectives:

Z_1	Maximum over all total weights $w^{(1)}$ of edges assigned to each agent, to be minimized
Z_2	Minimum over all total weights $w^{(2)}$ of edges assigned to each agent, to be maximized
Z_3	Chaos measure of the partition x on graph G, to be minimized

Table 3.1: Notation.

We now formulate the MIP $P_1(G)$.

$$
P_1(G) = \begin{cases}
\min \quad Z_1 & & (3.1a)\\
\max \quad Z_2 & & (3.1b)\\
\min \quad Z_3 & & (3.1c)\\
\text{s.t.} \quad \sum_{e\in E} w^{(1)}_{e,k} x_{e,k} \le Z_1 & \forall k \in K & (3.1d)\\
\sum_{e\in E} w^{(2)}_{e,k} x_{e,k} \ge Z_2 & \forall k \in K & (3.1e)\\
g(G,x) = Z_3 & & (3.1f)\\
\sum_{k\in K} x_{e,k} = 1 & \forall e \in E & (3.1g)\\
x_{e,k} \in \{0,1\} & \forall e \in E, k \in K & (3.1h)
\end{cases}
$$

Expression (3.1a) minimizes the first objective value Z_1, (3.1b) maximizes the second objective Z_2, (3.1c) minimizes the third objective Z_3.
Constraint (3.1d) sets the first objective equal to the maximum of total weights $w^{(1)}$ assigned to each agent. Constraint (3.1e) sets the second objective equal to the minimum of total weights $w^{(2)}$ assigned to each agent. Constraint (3.1f) sets the third objective equal to the chaos measure. Note that, depending on the desired chaos measure, one can choose the function $g(G,x)$. Constraint (3.1g) forces every edges to be assigned to exactly one agent. Lastly, constraint (3.1h) makes sure the partition variables $x_{e,k}$ are binary for every $e \in E$ and $k \in K$.

3.1. The chaos measure

In this thesis two versions of $g(G,x)$ will be discussed. The first version of $g(G,x)$ is the sum of the span of all nodes. To define this formally, we need some definitions. Let $N(v)$ be the set of edges adjacent to node v.

$$N(v) := \{e \in E : e \text{ adjacent to } v \text{ in } G\} \quad .$$

The span of a node v for an assignment x, labeled $\eta_v(x)$, is defined as the number of different agents assigned to edges in $N(v)$ in the assignment x.

$$\eta_v(x) := \sum_{k\in K} \mathbb{1}_{\exists e \in N(v): x_{e,k}=1} \quad .$$

Figure 2.3 shows an example graph with the span of each node. The function $g(G,x)$ now becomes

$$g(G,x) = \sum_{v\in V} \eta_v(x) \quad . \tag{3.2}$$

The second version of $g(G,x)$ discussed in this thesis is the number of "zones" of a partition x in graph G. Here, a zone is a subset of edges assigned to the same agent. We now give a definition of a zone.

Definition 1. *Let $G = (V,E)$ be a graph and K be a set of agents. Let $x_{e,k}$ be the edge-agent assignment. Then a subset $F \subseteq E$ is called a zone if and only if all of the following hold*

i) There exists a $k \in K$ such that $\forall e \in F$, we have $x_{e,k} = 1$.
ii) F is connected.
iii) $\nexists F' \supset F$ for which i) and ii) hold.

An example of a graph divided in zones is given in Figure 3.1.

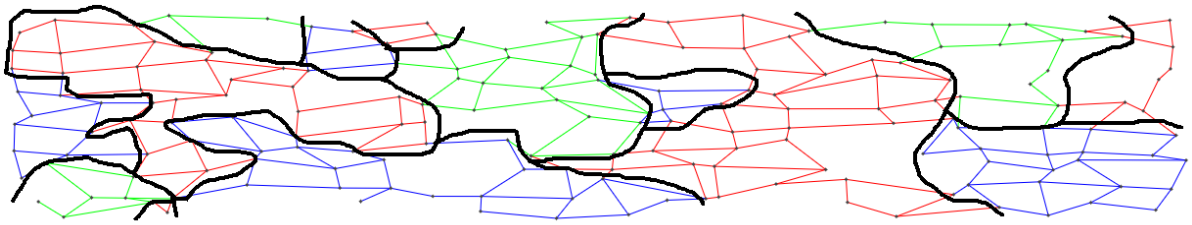


Figure 3.1: Zones for a certain edge-agent assignment.

4

Structural results

In this chapter we will show theoretical results for Unrelated Unconnected Multi-Constrained Graph Partitioning problem, or UUMCGP, whose mixed-integer program (MIP) $P_1(G)$ is described in Chapter 3. In Section 4.1 results are shown for specific cases of UUMCGP. In Section 4.2 we will show a theoretical result for the related Multi-Resource Generalized Assignment Problem.

4.1. Solving UUMCGP

In this section we will show that we can come up with good solutions in specific cases of UUMCGP. We will first show that UUMCGP is hard to solve in Subsection 4.1.1. We will then elaborate on four cases, define what we mean by "good" solutions and give supporting lemmas, programs and notation in Subsection 4.1.2. We will end by proving four theorems that give good solutions to UUMCGP in polynomial time in specific cases in Subsection 4.1.3.

4.1.1. Hardness of UUMCGP

Coming up with good solutions for UUMCGP in polynomial time is difficult. This is partly because UUMCGP is multi-objective. We want to optimize the niceness of a solution, balance the first resource among agents and balance the second resource among the agents. A solution to $P_1(G)$ can have one of the objectives optimized, while the other two objectives are far from their optimal value. In the case of debris management for example, do we prefer a solution with a fast operation time above a solution that yields good profit? How do we decide which objective is more important?

Suppose we do somehow decide on an objective Z_i that we think is the most important. Let us now only focus on that objective and ignore the other two completely. Let us denote the problem we are left with as $P_1(G, Z_i)$. By choosing only one objective instead of three, we hugely simplify the MIP $P_1(G)$. However, for $i = 1, 2$, our simplified MIP $P_1(G, Z_i)$ remains NP-hard. For $i = 3$ the optimal solution becomes trivial. This result is shown in Theorem 1.

Theorem 1. *Let $P_1(G, Z_i)$ denote the MIP $P_1(G)$ where we only optimize objective Z_i and ignore the other two objectives. $P_1(G, Z_i)$ is NP-hard for $i = 1, 2$ and trivial for $i = 3$.*

Proof. We prove this result separately for $i = 1$, $i = 2$ and $i = 3$.

Case 1: $i = 1$
In this case we only focus on optimizing Z_1. Recall that Z_1 denotes the maximum use of the first resource among all agents. We try to minimize Z_1. The MIP $P_1(G, Z_1)$ we end up with is $P_1(G)$ without the constraints (3.1b), (3.1c),(3.1e) and (3.1f). This is precisely the minimum makespan problem $R_m||C_{max}$ discussed in Chapter 2, which is NP-hard [9].

Case 2: $i = 2$
In this case we only focus on optimizing Z_2. Recall that Z_2 denotes the minimum use of the second resource among all agents. We try to maximize Z_2. The MIP $P_1(G, Z_2)$ we end up with is $P_1(G)$ without the

constraints (3.1a), (3.1c),(3.1d) and (3.1f). If we multiply both sides of Expression (3.1e) by -1 and change (3.1b) to $-\min(-Z_2)$, we again have the minimum makespan problem $R_m||C_{max}$, which is NP-hard [9].

Case 3: $i = 3$
In this case we only focus on optimizing Z_3. Remember that Z_3 equals the chaos measure that we try to minimize. The MIP $P_1(G, Z_3)$ we end up with is $P_1(G)$ without the constraints (3.1a), (3.1b),(3.1d) and (3.1e). In Section 3.1 we discussed the two versions of $g(G, x)$ considered in this thesis. For both versions of $g(G, x)$ the optimal solution of $P_1(G, Z_3)$ now is trivial: assign all edges to the same agent. When $g(G, x)$ is the sum of the span of the nodes, the objective value of Z_3 now becomes $|V|$, the number of nodes of graph G. When $g(G, x)$ is the number of zones, the objective value of Z_3 becomes 1. □

4.1.2. Cases of UUMCGP

The multi-objective character of UUMCGP and the result of Theorem 1 make solving $P_1(G)$ hard or impossible in polynomial time. However, when we simplify $P_1(G)$ by forcing its parameters to be of a certain form, we can still come up with good solutions. The next subsection, 4.1.3, provides 4 theorems that do exactly that. Each theorem shows that we can come up with a good solution in polynomial time in a specific case. In this subsection we will first describe all 4 of the specific cases and their characteristics. We will then explain what we mean exactly with a "good" solution. We will end by proving some lemmas that will serve as support for the 4 theorems.

Four cases of UUMCGP
The MIP $P_1(G)$ of UUMCGP can be simplified by putting restrictions on its parameters. One of these parameters is the structure of the graph G. In all four theorems of this section we will put restrictions on the structure of the graph concerning the vertex connectivity. We will now give the definition of this vertex connectivity. Examples of vertex connectivity are given in Figures 4.1 and 4.2.

Definition 2. *Let $G = (V, E)$ be an undirected graph. G is l-vertex-connected when for every subset $V' \subset V$ with $|V'| < l$, the graph $G[V \setminus V']$ is connected.*

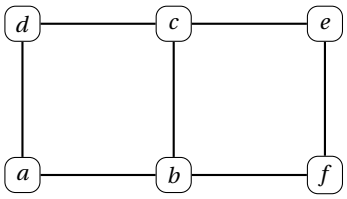


Figure 4.1: This graph is 2-vertex-connected. Since we can remove any vertex and the graph will remain connected. It is not 3-vertex-connected, since removing vertex c and b makes the graph unconnected.

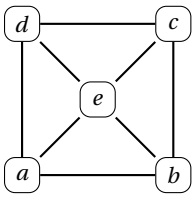


Figure 4.2: This graph is 3-vertex-connected. Since we can remove any two vertices and the graph will remain connected. It is not 4-vertex-connected, since removing vertices d, e and b makes the graph disconnected.

Other parameters of $P_1(G)$ are the number of agents r, the weight values $w_{e,k}^{(i)}$ and the chaos measure $g(G, x)$. Theorems 2-5 will all consider $P_1(G)$ with different values for these parameters. We will now describe these values for each theorem. An overview of the Theorems and their corresponding parameter choices is given in Table 4.1.

In Theorem 2, we assume that G is a 2-vertex-connected graph. We assume there are only 2 agents, (i.e. $r = 2$). All weight values $w_{e,k}^{(i)}$ are of the form $w_{e,k}^{(i)} = w_e C_k^{(i)}$. Here w_e is a positive integer for every $e \in E$ and $C_k^{(i)}$ is a

positive real number for $k = 1,2$ and $i = 1,2$. In the debris management problem w_e can be interpreted as the amount of debris on street e in units of debris. $C_k^{(1)}$ represents the time it takes contractor k to clean one unit of debris and $C_k^{(2)}$ represents the profit contractor k makes cleaning one unit of debris.

In Theorem 3, we assume that G is a $\min(r,4)$-vertex planar connected graph. There is no restriction on the number of agents r. All weight values $w_{e,k}^{(i)}$ are of the form $w_{e,k}^{(i)} = C_k^{(i)}$. Here $C_k^{(i)}$ is a positive real number for $k \in K$ and $i = 1,2$. This can be interpreted as the debris management problem with one unit of debris on every street. Again, $C_k^{(1)}$ represents the time it takes contractor k to clean one unit of debris and $C_k^{(2)}$ the profit contractor k makes cleaning one unit of debris.

In Theorem 4, we again assume that G is a $\min(r,4)$-vertex planar connected graph. Again, there are no restrictions on the number of agents r. All weight values $w_{e,k}^{(i)}$ are now drawn from a distribution $\chi_k^{(i)}$ with mean $\mu_k^{(i)}$. In the debris management problem this again reflects one unit of debris on every street. However, the time and profit per unit of debris now come from a random distribution and they are allowed to differ for every edge. Contractor 1, for example, can clean one unit of debris in one hour on a certain street, but it might need two hours for one unit of debris on another.

In Theorem 5, we again assume that G is a $\min(r,4)$-vertex planar connected graph. Again, there are no restrictions on the number of agents r. The weights in this case represent the most realistic case of the debris management problem. All weight values $w_{e,k}^{(i)}$ are of the form $w_{e,k}^{(i)} = w_e^{(i)} C_k$, where the parameters $w_e^{(1)}$ ($w_e^{(2)}$) are identically distributed variables from distribution χ^1 (χ^2). The parameters C_k are positive real numbers for all $k \in K$. We can view $w_e^{(1)}$ as the time it takes one employee to clean street e and $w_e^{(2)}$ as the profit made when one employee cleans street e. C_k can the be viewed as one divided by the number of employees of contractor k. Hiring more employees now results in shorter cleaning times but lower profits, due to salaries for example.

Theorem	**G is**	$r =$	$w_{e,k}^{(i)}$	**g(G,x)**
2	2-vertex-connected	2	$w_e C_k^{(i)}$	number of zones
3	$\min(r,4)$-vertex-connected	no restr.	$C_k^{(i)}$	number of zones
4	$\min(r,4)$-vertex-connected	no restr.	$\sim \chi_k^{(i)}$	number of zones
5	$\min(r,4)$-vertex-connected	no restr.	$w_e^{(i)} C_k$, where $w_e^{(i)} \sim \chi^{(i)}$	number of zones

Table 4.1: Overview of restrictions on the parameters in $P_1(G)$ in Theorems 2 - 5.

For all Theorems 2-5 we take for our function $g(G,x)$ the number of zones as defined in Definition 1. The theorems will show that we can come up with good solutions to $P_1(G)$ in polynomial time. We will now explain what we mean by a "good" solution.

Good solution

What does it mean for a solution to be good in a multi-objective problem like UUMCGP?

Let us start with objective value Z_3, which equals the number of zones in our 4 specific cases. When can we say that Z_3 has a good value? Theorem 1 shows that the optimal value for Z_3 is 1. This can be obtained by assigning all edges to one agent. This solution, however, is non-realistic, because it results in very bad Z_1 and Z_2 values. Every agent needs to be assigned at least one or more edges to obtain reasonable values for the Z_1 and Z_2 objective. Hence, in every realistic solution, each agent has at least one zone assigned to it. The number of zones is therefore equal to or greater than the number of agents. A good value for Z_3 is therefore the number of agents, r.

Can we do something similar for Z_1 and Z_2? When do we consider these objectives to have good values? Remember that Z_1 needs to be minimized and Z_2 needs to be maximized. With the assumption that we consider both objectives equally important, we can then replace the two objectives by the single objective $Z_{1,2} := \frac{Z_2}{Z_1}$, which we will try to maximize. Let $Z_{1,2}^{OPT}$ denote the optimal value of $Z_{1,2}$. In Theorems 2-4 we will see that we can find solutions in polynomial time whose $Z_{1,2}$ objective is arbitrarily close to its optimal value $Z_{1,2}^{OPT}$

for a large enough size of the edge set, $|E|$. Or put differently, for every $\delta > 0$, there exists a $n \in \mathbb{N}$ such that $P_1(G)$ on a graph with $|E| = n$ has a solution with $|Z_{1,2}^{OPT} - Z_{1,2}| < \delta$.

In Theorem 5 we do something different. Here we do not maximize the objective value $Z_{1,2}$, but we stick with our original goal: minimize Z_1 and maximize Z_2. Let Z_1^{OPT} and Z_2^{OPT} denote their optimal values. We will show that in a specific case of $P_1(G)$ we can find a solution that has Z_1 and Z_2 values arbitrarily close to their optimal value. Here, arbitrarily close is defined different: for every $\delta > 0$, there exists a $n \in \mathbb{N}$ such that $P_1(G)$ on a graph with $|E| = n$ has a solution with $\frac{|Z_1^{OPT} - Z_1|}{Z_1^{OPT}} < \delta$ and $\frac{|Z_2^{OPT} - Z_2|}{Z_2^{OPT}} < \delta$

In Table 4.2 an overview is given of the values of the objective values in the solutions found in Theorems 2-5.

Theorem	Z_1	Z_2	Z_3
2	$Z_{1,2} \to Z_{1,2}^{OPT}$		2
3	$Z_{1,2} \to Z_{1,2}^{OPT}$		r
4	$Z_{1,2} \to Z_{1,2}^{OPT}$		r
5	$Z_1 \to Z_1^{OPT}$	$Z_2 \to Z_2^{OPT}$	r

Table 4.2: Overview of objective values of solutions we find in Theorems 2 - 5.

Supporting lemmas, programs and notation
Before proving the four main theorems, we will give supporting lemmas, programs and notation that will be used while proving the theorems. An overview of the programs and notation introduced here is given in Table 4.3.

Let us start with an important piece of notation. Let P be a program with a feasible solution x. With $Z(P,x)$ we denote the value of objective value Z in program P of the feasible solution x.

The first program we define is P_2. With P_2 we denote the MIP obtained by ignoring the graph structure of $P_1(G)$. Or, put differently, P_2 is $P_1(G)$ without Expression (3.1c) and (3.1f).

We can simplify program P_2 to obtain program $\Pi_1(r)$. To do this, we replace the binary variables $x_{e,k}$ in P_2 by $S_k := \frac{\sum_{e \in E} x_{e,k}}{|E|}$. S_k can be seen as the percentage of edges assigned to agent k. We also introduce the continuous increasing functions $f_k^{(i)} : [0,1] \to \mathbb{R}$ for $i = 1,2$ and for all $k \in K$. $f_k^{(1)}(S_k)$ now represents the amount of the first resource that is used by agent k when we assign S_k% of the edges to it. $f_k^{(2)}(S_k)$ represents the amount of the second resource that is used by agent k when we assign S_k% of the edges to it. The question $\Pi_1(r)$ answers is as follows: what percentage of the edges do we give to each agent to balance the use of the first and second resource?

$$\Pi_1(r) = \begin{cases} \min Z_1 & (4.1a) \\ \max Z_2 & (4.1b) \\ s.t. \quad f_k^{(1)}(S_k) \le Z_1 \quad \forall k \in \{1,\dots,r\} & (4.1c) \\ f_k^{(2)}(S_k) \ge Z_2 \quad \forall k \in \{1,\dots,r\} & (4.1d) \\ \sum_{k=1}^{r} S_k = 1 & (4.1e) \\ S_k \ge 0 \quad \forall k \in \{1,\dots,r\} & (4.1f) \end{cases}$$

$\Pi_1(r)$ differs greatly from $P_1(G)$ and one might wonder how it might be useful for finding good solutions for $P_1(G)$. The reason we defined $\Pi_1(r)$ is because all its Pareto optimal solutions are of a certain form. This result turns out to be useful for proving our theorems. It is proven in Lemma 1 and states:

$$S \text{ is Pareto Optimal} \iff Z_1(\Pi_1(r), S) = \max_{k \in \{1,\dots,r\}} [f_k^{(1)}(f_k^{(2)-1}(Z_2(\Pi_1(r), S))] \tag{4.2}$$

This can be explained best with an example. Suppose $r = 2$ and let $f_1^{(1)}(S_1) = S_1$, $f_2^{(1)}(S_2) = (S_2)^2$, $f_1^{(2)}(S_1) =$

$0.5S_1$, $f_2^{(2)}(S_2) = 3S_2$. When we denote $Z_i(\Pi_1(r), S))$ with Z_i, we get

$$f_1^{(1)}(f_1^{(2)-1}(Z_2)) = f_1^{(1)}(2Z_2) = 2Z_2$$

and

$$f_2^{(1)}(f_2^{(2)-1}(Z_2)) = f_2^{(1)}(\frac{1}{3}Z_2) = \frac{1}{9}Z_2^2.$$

Hence, a feasible solution S is Pareto optimal if we have, for the objective values Z_1 and Z_2, that

$$Z_1 = \max[2Z_2, \frac{1}{9}Z_2^2].$$

We will now prove Statement (4.2).

Lemma 1. *Consider* $\Pi_1(r)$ *where the* $f_k^{(i)} : [0,1] \to \mathbb{R}$ *are increasing functions for* $i = 1,2$ *and all* $k \in K$. *Then the following holds:*

$$S \text{ is Pareto Optimal} \iff Z_1(\Pi_1(r), S) = \max_{k\in\{1,\dots,r\}}[f_k^{(1)}(f_k^{(2)-1}(Z_2(\Pi_1(r), S))].$$

Proof. Let r be given. In this proof we denote $Z_i(\Pi_1(r), S)$ by $Z_i(S)$ for $i = 1,2$.
"$\Longrightarrow$"
Let S be a Pareto optimal solution to $\Pi_1(r)$. This proof consists of two steps:

1: We show that $Z_1(S) > \max_{k\in\{1,\dots,r\}}[f_k^{(1)}(f_k^{(2)-1}(Z_2(S))]$ leads to a contradiction.

2: We show that $Z_1(S) < \max_{k\in\{1,\dots,r\}}[f_k^{(1)}(f_k^{(2)-1}(Z_2(S))]$ leads to a contradiction.

The required result than automatically follows.

Step 1: Suppose

$$Z_1(S) > \max_{k\in\{1,\dots,r\}}[f_k^{(1)}(f_k^{(2)-1}(Z_2(S))]. \tag{4.3}$$

We then have

$$Z_1(S) > f_k^{(1)}(f_k^{(2)-1}(Z_2(S)) \quad \forall k \qquad \Rightarrow \tag{4.4}$$

$$f_k^{(1)-1}(Z_1(S)) > f_k^{(2)-1}(Z_2(S)) \quad \forall k. \tag{4.5}$$

We will now pick another solution $S' = [S'_1, \dots, S'_r]$ such that

$$f_k^{(1)-1}(Z_1(S)) > S'_k > f_k^{(2)-1}(Z_2(S)) \quad \forall k.$$

We will show that S' is feasible and more optimal than S. This leads to a contradiction, since S was Pareto optimal. We will first show it is feasible by showing that we can choose S' in such a way that $\sum_k S'_k = 1$. Note that it is sufficient to show

$$\sum_k f_k^{(2)-1}(Z_2(S)) \le 1 \quad \text{and} \tag{4.6}$$

$$\sum_k f_k^{(1)-1}(Z_2(S)) \ge 1. \tag{4.7}$$

We will prove Expression (4.6). Let $S_p = [S_{p,1}, \dots, S_{p,r}]$ be the solution that optimizes the objective value Z_2. We have

$$\begin{aligned} Z_2(S_p) &\ge Z_2(S) && \Rightarrow \\ f_k^{(2)}(S_{p,k}) &\ge Z_2(S) \quad \forall k && \Rightarrow \\ S_{p,k} &\ge f_k^{(2)-1}(Z_2(S)) \quad \forall k && \Rightarrow \\ \sum_k S_{p,k} &\ge \sum_k f_k^{(2)-1}(Z_2(S)). \end{aligned}$$

But S_p is a feasible solution and therefore has $\sum_k S_{p,k} = 1$. That gives Expression (4.6). Expression (4.7) has a similar proof that we leave out here. We conclude that S' can be chosen such that it is a feasible solution to $\Pi_1(r)$.

It remains to show that S' dominates S. Recall that we chose S' such that

$$f_k^{(1)-1}(Z_1(S)) > S'_k > f_k^{(2)-1}(Z_2(S)) \quad \forall k.$$

This gives

$$\begin{aligned} & f_k^{(1)}(S'_k) < Z_1(S) \quad \forall k && \Rightarrow \\ & Z_1(S') < Z_1(S). \end{aligned}$$

And similarly,

$$\begin{aligned} & f_k^{(2)}(S'_k) > Z_2(S) \quad \forall k && \Rightarrow \\ & Z_2(S') > Z_2(S). \end{aligned}$$

Hence S' is a feasible solution to $\Pi_1(r)$ and dominates S. Therefore S can not be Pareto optimal. This is a contradiction. We conclude that Expression (4.3) is false.

Step 2: Suppose

$$Z_1(S) < \max_{k\in\{1,\dots,r\}} [f_k^{(1)}(f_k^{(2)-1}(Z_2(S))]. \tag{4.8}$$

This gives for some k

$$f_k^{(1)-1}(Z_1(S)) < f_k^{(2)-1}(Z_2(S)).$$

But now we have for this k that

$$S_k = f_k^{(1)-1}[f_k^{(1)}(S_k)] \leq f_k^{(1)-1}[Z_1(S)] < f_k^{(2)-1}[Z_2(S)] \leq f_k^{(2)-1}[f_k^{(2)}(S_k)] = S_k.$$

This is a contradiction. We conclude that Expression (4.8) must be false.

Since Expressions (4.3) and (4.8) are false, we conclude that

$$Z_1(S) = \max_{k\in\{1,\dots,r\}} [f_k^{(1)}(f_k^{(2)-1}(Z_2(S))].$$

"$\Longleftarrow$"
Let S be a feasible solution to $\Pi_1(r)$ that has

$$Z_1(S) = \max_{k\in\{1,\dots,r\}} [f_k^{(1)}(f_k^{(2)-1}(Z_2(S))]. \tag{4.9}$$

We have to show that S is Pareto optimal. Suppose for contradiction that there exists a feasible solution $S' = [S'_1, \dots, S'_k]$ that dominates S. Hence, we have for this S' that $Z_1(S') < Z_1(S)$ and $Z_2(S') > Z_2(S)$. This gives

$$\begin{aligned} & Z_2(S') > Z_2(S) && \Rightarrow \\ & f_k^{(2)}(S'_k) > Z_2(S) \quad \forall k && \Rightarrow \\ & S'_k > f_k^{(2)-1}[Z_2(S)] \quad \forall k. && \end{aligned} \tag{4.10}$$

Similarly, we have

$$S'_k < f_k^{(1)-1}[Z_1(S)]. \tag{4.11}$$

Moreover, we can rewrite Expression (4.9) to obtain for some k

$$f_k^{(1)-1}[Z_1(S)] \leq f_k^{(2)-1}[Z_2(S)]. \tag{4.12}$$

When we now combine Expressions (4.10)-(4.12), we obtain for some k

$$S'_k < f_k^{(1)-1}[Z_1(S)] \leq f_k^{(2)-1}[Z_2(S)] < S'_k.$$

This is a contradiction. Hence, there is no solution S' to $\Pi_1(r)$ that dominates S. We conclude that S is Pareto optimal. □

We also define the programs $\Pi_2(r)$ and $\Pi_3(r)$. They are specific cases of $\Pi_1(r)$. In $\Pi_2(r)$ the functions $f_k^{(i)}(S_k)$ are set to be $f_k^{(i)}(S_k) := C_k^{(i)} S_k$. In $\Pi_3(r)$ the functions $f_k^{(i)}(S_k)$ are set to be $f_k^{(i)}(S_k) := \mu_k^{(i)} S_k$.

In Theorem 4 and 5 we draw the weights from random distributions. A lemma we use to deal with these random numbers is Lemma 2 about the law of large numbers. Due to this law we know that the mean of the sum of i.i.d. random variables from a distribution converges to the mean of that distribution. In Lemma 2 we show the rate of this convergence.

Lemma 2. *Let $x_j \sim \chi$ be i.i.d. random variables with finite expected value μ and standard deviation σ. Then the following holds for $\epsilon > 0$ and large n*

$$\left| \frac{\sum_{j=1}^{n} x_j}{n} - \mu \right| \leq n^{-0.5+\epsilon}.$$

Proof. Chebyshev's inequality states the following for a real number $k > 0$ and a random variable X with finite expected value $\mu(X)$ and with finite non-zero standard deviation $\sigma(X)$:

$$Pr(|X - \mu(X)| \geq k\sigma(X)) \leq \frac{1}{k^2}.$$

Choosing $k := \frac{\epsilon'}{\sigma(X)}$ for some $\epsilon' > 0$ we obtain

$$Pr(|X - \mu(X)| \geq \epsilon') \leq \frac{\sigma(X)^2}{\epsilon'^2}. \tag{4.13}$$

We take $X := \frac{\sum_{j=1}^{n} x_j}{n}$. Note that $\mu(X) = \mu$ and $\sigma(X) = \frac{\sigma}{\sqrt{n}}$ We plug these values in Inequality (4.13) to obtain

$$Pr\left(\left| \frac{\sum_{j=1}^{n} x_j}{n} - \mu \right| \geq \epsilon' \right) \leq \frac{\sigma^2}{n\epsilon'^2}.$$

If we now choose $\epsilon' := n^{-0.5+\epsilon}$ for some $\epsilon > 0$ we get

$$Pr\left(\left| \frac{\sum_{j=1}^{n} x_j}{n} - \mu \right| \geq n^{-0.5+\epsilon} \right) \leq \frac{\sigma^2}{n^{2\epsilon}}.$$

Now, for large enough n

$$Pr\left(\left| \frac{\sum_{j=1}^{n} x_j}{n} - \mu \right| \geq n^{-0.5+\epsilon} \right) \leq 0, \text{ which implies}$$

$$\left| \frac{\sum_{j=1}^{n} x_j}{n} - \mu \right| \leq n^{-0.5+\epsilon}.$$

□

The last supportive notion we will state is Lemma 3. It shows a results regarding the vertex connectivity of a graph, which is defined in Definition 2. Győri and Lovász already showed separately that the vertex set V of an r-vertex-connected graph can be partitioned in r disjoint connected subsets [16],[23]. Lemma 3 shows a similar result for a partitioning of the edge set E. It states that when a graph G is $\min(4,r)$-vertex-connected, we can partition its edges into r disjoint connected subsets. A result we use when proving the 4 main theorems of this chapter. When A is a subset of the vertices or edges, we will use $G[A]$ to denote the subgraph of G induced by A.

Lemma 3. *Let $G=(V,E)$ be an undirected $\min(4,r)$-vertex-connected planar graph. Let $A_1,A_2,\ldots,A_r \in \mathbb{N}$ be a sequence of integers such that $\sum_j A_j = |E|$. Then there exists an edge-partitioning x of E into disjoint subsets $E_1,\ldots,E_r$ such that $G[E_j]$ is connected for all $j \in \{1,\ldots,r\}$ and $|E_j| = A_j$ for all $j \in \{1,2,\ldots,r\}$.*

Proof. We prove this lemma separately for $r=1$, $r=2,3$ and $r \geq 4$.
Case 1: $r=1$: We need to partition the edge set E into one subset E_1. This is a trivial case. The solution is $E_1 = E$.

Case 2: $r=2,3$: We have to show, for $r=2,3$, that when a graph is r-vertex-connected, we can partition the edge set into r disjoint connected subsets of arbitrary size. We will show that this is true for all $r=2,3,4,\ldots$. We use the following steps:

1. We use G to construct the dual graph G'.
2. G' is r-vertex-connected.
3. There exists an arbitrary vertex partition in G'.
4. This vertex partition in G' corresponds to an edge partition in G.

Step 1: Let $G=(V,E)$ be the original graph. The dual graph $G'=(V',E')$ now is defined by $V' := E$ and $E' := \{(v,w) \in V' \times V' : \text{edges } v \text{ and } w \text{ share a vertex in } G\}$. In Figure 4.3 such a dual graph is constructed.

(a) Original graph G. (b) Dual graph G'.

Figure 4.3: Constructing a dual graph G'.

Step 2: We now show that when G is r-vertex-connected, its dual G', is also r-vertex-connected. We use Menger's theorem [27], that states that

$$G \text{ is } r\text{-vertex-connected} \iff \text{every pair of vertices has } r \text{ internally vertex-disjoint paths in between.}$$

We show that for every $v,w \in V'$ there are r internally vertex-disjoint paths between them in G'. v corresponds to an edge (v_1,v_2) and w corresponds to an edge (w_1,w_2) in the original graph G. Since G is r-vertex-connected, we know that there are r internally vertex-disjoint paths between v_1 and w_1. But, vertex-disjoint paths are also edge-disjoint. And every edge-disjoint path in G again corresponds to a vertex-disjoint path in G'. Hence there are r vertex-disjoint path in G' between v and w. We use Menger's theorem to conclude that G' is also r-vertex-connected.

Step 3: We now show there exists a vertex partition of V' into disjoint subsets $V'_1,V'_2,\ldots,V'_r$, where $|V'_j| = A_j$ and $G[V'_i]$ is connected for $j=1,2,\ldots,r$. The Győri-Lovász theorem [16] states exactly that.

Step 4: We can now construct the required edge partition in the original graph G by setting the vertices in the vertex partition in G' to be the edges in the edge partition in G. Or, $E_j := V'_j$ for $j=1,2,\ldots,r$. Clearly, we have $|E_j| = |V'_j| = A_j$. It remains to show that for all $j=1,2,\ldots,r$ the subgraph induced by E_j is connected. Let $e,f \in E_j$. Hence, $e,f \in V'_j$. Since V'_j is connected there exists a path in G' connecting e and f. But this path in

$G'[V'_j]$ corresponds to a path in $G[E_j]$. Hence, there exist a path between e and f in $G[E_j]$. We conclude that $G[E_j]$ is connected for all $j = 1,2,\ldots,r$.

Case 4: $r \geq 4$: We will show that in a 4-vertex-connected graph , for every $r \geq 4$ and $A_1,\ldots,A_r$ positive integers that sum up to $|E|$, there always exists a partition of the edge set E into disjoint subsets $E_1,E_2,\ldots,E_r$ such that $|E_j| = A_j$ and $G[E_j]$ connected for all $j = 1,2,\ldots,4$.

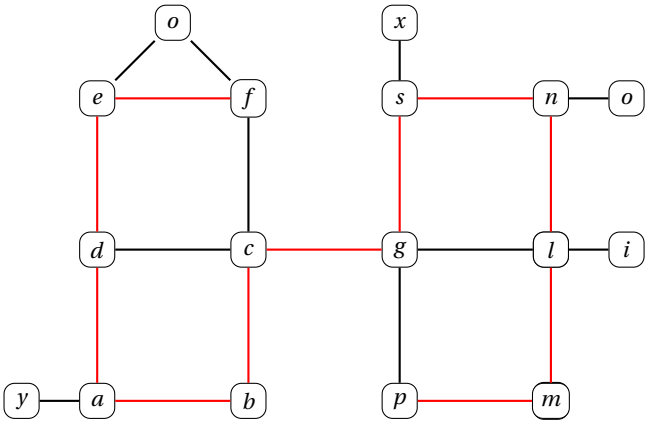
Tutte showed in 1956 that r-connected planar graphs have a Hamiltonian path for $r \geq 4$ [32]. A Hamiltonian path is a path that visits every vertex exactly once. It remains to show that the theorem holds for graphs with a Hamiltonian path.

We will show something stronger: the theorem holds when there exists a path P such that for every edge $e \in E$, we have that $e \in P$ or e is connected with a vertex $v \in P$. We call such a path P a Caterpillar path. Clearly, a Hamiltonian path is a Caterpillar path.

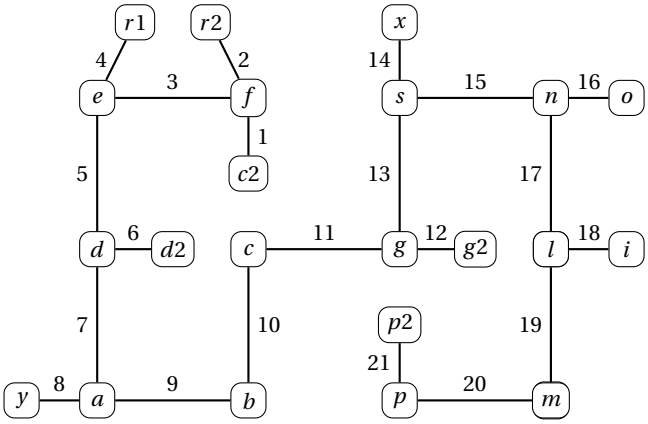
We start with a graph with a Caterpillar path whose edge set we have to partition into disjoint subsets E_1, E_2, $\ldots$, E_r with sizes $A_1,A_2,\ldots,A_r$, such that $G[E_j]$ is connected for $j = 1,2,\ldots,r$. A graph with a Caterpillar path is shown in Figure 4.4a. For every edge that is not in the Caterpillar path, we duplicate one end vertex. This gives us a tree whose edges we can label. The splitting and labeling is shown in Figure 4.4b. We now partition the edges set as follows:

$$
\begin{aligned}
E_1 &= \{1,2,\ldots,A_1\} \\
E_2 &= \{A_1+1, A_1+2,\ldots,A_1+A_2\} \\
&\vdots \\
E_j &= \left\{\sum_{i=1}^{j-1} A_i + 1,\ldots,\sum_{i=1}^{j} A_i\right\} \\
&\vdots \\
E_r &= \left\{\sum_{i=1}^{r-1} A_i + 1,\ldots,|E|\right\}
\end{aligned}
$$

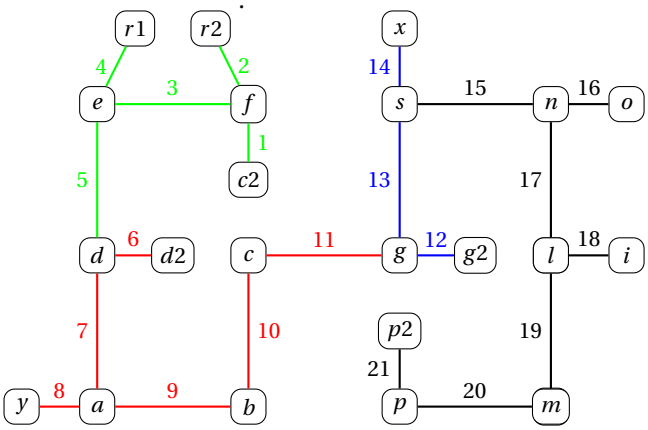
This partitioning is shown in Figure 4.4c. We then stitch the broken vertices back together, see Figure 4.4d. It is easy to see that this gives the required partition. □



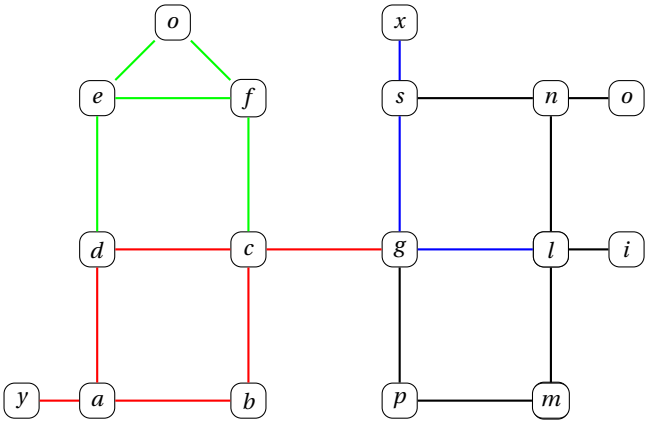
(a) A graph with a Caterpillar path in red. Note that every edge is either contained in the caterpillar or connected to the caterpillar.



(b) We disconnect one vertex of all edges that are not contained in the Caterpillar path. We label the edges of the resulting graph.



(c) We color the edges according to the labeling.



(d) We stitch the disconnected vertices back together and obtain an edge partition.

Figure 4.4: An example graph whose edge set is partitioned in subsets of sizes $A_1 = 5$, $A_2 = 6$, $A_3 = 3$, $A_4 = 7$.

In this subsection we have introduced notation that we will use to prove the four theorems in the next section. In Table 4.3 an overview of this notation is given.

Programs:

$P_1(G)$	Program defined by Expression (3.1)
P_2	$P_1(G)$ without Expression (3.1c) and (3.1f)
$\Pi_1(r)$	Program defined in Expression (4.1)
$\Pi_2(r)$	$\Pi_1(r)$ with $f_k^{(i)}(S_k) = C_k^{(i)} S_k$
$\Pi_3(r)$	$\Pi_1(r)$ with $f_k^{(1)}(S_k) = \mu_k^{(1)} S_k$ and $f_k^{(2)}(S_k) = \mu_k^{(2)} S_k$

Decision variables:

$S = [S_1, \ldots, S_r]$	Set of decision variables for $\Pi_1(r), \Pi_2(r)$ and $\Pi_3(r)$
$x_{e,k}$	Decision variables for $P_1(G)$, P_2, as defined in Table 3.1

Other Notation:

$Z(P,x)$	Value of objective Z in program P at the feasible solution x
$Z_{1,2}$	$\frac{Z_2}{Z_1}$
$Z_{1,2}^{OPT}$	Optimal maximum value of $Z_{1,2}$ in $P_1(G)$
$Z_i^{OPT}(P)$	Optimal value of Z_i in problem P

Table 4.3: An overview of the programs, decision variables and notation used to prove Theorems 2-5.

4.1.3. Theorems

In the previous subsection we have described all four cases. Moreover, we explain what we mean by a good solution and gave supporting lemmas, programs and notation. This section will contain the theorems with their proofs.

Theorem 2. *We consider $P_1(G)$ with the following restrictions.*

G is a 2-vertex-connected graph.
There are two agents, i.e. $K = \{1,2\}$.
$w_{e,k}^{(i)} := w_e C_k^{(i)}$ for $i = 1,2$, $k = 1,2$, $e \in E$, where $w_e \in \mathbb{N}$ and $C_k^{(i)} \in \mathbb{R}_+$.
$g(G,x) :=$ the number of zones in the partition x on graph G.

Then, there exists a feasible solution x with

- $Z_3 = 2$.
- *For every $\delta > 0$, there exists an $n \in \mathbb{N}$ such that $P_1(G)$ on a graph with $|E| = n$ has a solution with $|Z_{1,2}^{OPT} - Z_{1,2}| < \delta$. Moreover, $Z_{1,2}^{OPT} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}$.*

Proof. The proof consists of the following steps.

1. We relax problem $P_1(G)$ to problem $P_1(G')$ by splitting the edges of graph G to obtain a graph G' with unit edge weights.
2. We relax problem $P_1(G')$ to P_2 by leaving out the graph structure.
3. We rewrite and relax problem P_2 to obtain the continuous problem $\Pi_2(2)$.
4. We prove that all solutions of $\Pi_2(2)$ on the Pareto Front have $\frac{Z_2(\Pi_2(2),S)}{Z_1(\Pi_2(2),S)} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}$.
5. We round the solutions of $\Pi_2(2)$ to obtain solutions for problem P_2.

6. We show that every solution to P_2 corresponds to a solution of $P_1(G')$.
7. We round the obtained partition for graph G' to find a solution to $P_1(G)$.
8. We combine all results and conclude.

Step 1: We first split every edge e of G into $w(e)$ equal parts. For each $e \in E$, we denote the edges obtained from splitting e with F_e. After splitting, we obtain the graph $G' = (V', E')$. Note that $E' = \cup_{e \in E} F_e$. Moreover, note that G' is a 2-connected undirected graph with unit edge weights. Note that any solution to $P_1(G)$ can be written as a solution to $P_1(G')$. This does not work the other way around. $P_1(G')$ can be seen as a relaxation of $P_1(G)$, where we allow every edge to be assigned to more than one agent.

Step 2: To solve problem $P_1(G')$, we first ignore the graph structure and chaos measure to obtain problem P_2. We do this so we can rewrite and relax the remaining problem in Step 3 and apply Lemma 1 in Step 4. We define P_2 by $P_1(G')$ without the objective Z_3 and constraint (3.1f). Clearly, P_2 is a relaxation of $P_1(G')$.

Step 3: We will now relax and rewrite P_2 to obtain $\Pi_2(2)$. We relax the integer constraint (3.1h) of P_2. Moreover, we start by rewriting constraint (3.1d) as follows

$$\begin{aligned}
&\sum_{e \in E'} w_{e,k}^{(1)} x_{e,k} \leq Z_1 && \forall k \in K && \Rightarrow \\
&C_k^{(1)} \sum_{e \in E'} x_{e,k} \leq Z_1 && \forall k \in K && \Rightarrow \\
&C_k^{(1)} \frac{\sum_{e \in E'} x_{e,k}}{|E'|} \leq \frac{Z_1}{|E'|} && \forall k \in K &&
\end{aligned}$$

Let S_k now denote the percentage of edges assigned to agent k in a solution to P_2. Or $S_k := \frac{\sum_{e \in E'} x_{e,k}}{|E'|}$, where integrality on $x_{e,k}$ is relaxed. We then obtain the following constraint.

$$C_k^{(1)} S_k \leq \frac{Z_1}{|E'|} \qquad \forall k \in K$$

Similarly we can rewrite constraint (3.1e) to

$$C_k^{(2)} S_k \geq \frac{Z_2}{|E'|} \quad \forall k \in K.$$

Finally, we can rewrite constraint (3.1g) as follows:

$$\begin{aligned}
&\sum_{k \in K} x_{e,k} = 1 && \forall e \in E' && \Rightarrow \\
&\sum_{k \in K} \sum_{e \in E'} x_{e,k} = |E'| && && \Rightarrow \\
&\sum_{k \in K} \frac{\sum_{e \in E'} x_{e,k}}{|E'|} = 1 && && \Rightarrow \\
&\sum_{k \in K} S_k = 1 && &&
\end{aligned}$$

Hence, $\Pi_2(2)$ is a relaxation of P_2.
In what follows we will compute all solutions on the Pareto Front of problem $\Pi_2(2)$. We will then round the solutions to obtain a solution to problem P_2. Note that this solution is unrelated to the graph structure and chaos measure. In Step 6 we will show however that we can transform a solution to P_2 to a solution to $P_1(G')$. In Step 7 we show that the solution to $P_1(G')$ can then be transformed to a solution to $P_1(G)$.

Step 4: We will prove that every solution on the Pareto Front of $\Pi_2(2)$ will satisfy $\frac{Z_2(\Pi_2(2),S)}{Z_1(\Pi_2(2),S)} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}$.
Note that $\Pi_2(2)$ is a special case of $\Pi_1(2)$ with $f_k^{(1)}(S_k) = C_k^{(1)} S_k$ and $f_k^{(2)}(S_k) = C_k^{(2)} S_k$. Hence, using Lemma 1,

we have for all solutions S on the Pareto Front of $\Pi_2(2)$, that

$$Z_1(\Pi_2(2),S) = \max_{k\in K}[f_k^{(1)}(f_k^{(2)-1}(Z_2(\Pi_2(2),S)))] \quad \Rightarrow$$

$$Z_1(\Pi_2(2),S) = \max_{k\in K}[\frac{C_k^{(1)}}{C_k^{(2)}} Z_2(\Pi_2(2),S)] \quad \Rightarrow$$

$$\frac{Z_2(\Pi_2(2),S)}{Z_1(\Pi_2(2),S)} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}.$$

Step 5: Let $S = [S_1, S_2]$ be a solution of $\Pi_2(2)$. We will show in this step that there exists a solution x'' of P_2 with objective values $Z_1(P_2,x'') \leq Z_1(\Pi_2(2),S)|E'| + 0.5\max_k(C_k^{(1)})$ and $Z_2(P_2,x'') \geq Z_2(\Pi_2(2),S)|E'| - 0.5\max_k(C_k^{(2)})$. To obtain this solution x'' we simply assign the first S_1% of the edges to agent 1 and the last S_2% of the edges to agent 2. There may be one edge that falls both in the first S_1% and the last S_2% of the edges. We assign that edge using a rounding technique. The process is depicted in Figure 4.5.

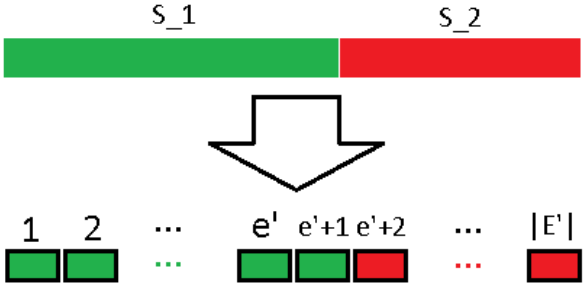


Figure 4.5: Going from solution $S = [S_1, S_2]$ to solution x''

Let $e' := \lfloor S_1|E'| \rfloor$, where $\lfloor q \rfloor$ denotes the integer part of a real number q. For $k = 1$, we choose

$$x''_{e,1} = \begin{cases} 1 & \text{if } e \in \{1,2,\ldots,e'\} \\ 0 & \text{otherwise.} \end{cases}$$

For $k = 2$, we choose

$$x''_{e,2} = \begin{cases} 1 & \text{if } e \in \{e'+2, e'+3, \ldots, |E'|\} \\ 0 & \text{otherwise.} \end{cases}$$

We are left with one unassigned edge $e = e' + 1$. We assign this element as follows

$$x''_{e',k} = \begin{cases} 1 & \text{if } dec(|E'|S_k) \geq 0.5 \\ 0 & \text{otherwise.} \end{cases}$$

Here, $dec(a) := a - \lfloor a \rfloor$, ie., the decimal part of a.
Suppose that $dec(|E'|S_1) \geq 0.5$. In this case, we have for constraint (3.1d) with $k = 1$

$$\begin{aligned} \sum_{e\in E'} w_{e,1}^{(1)} x''_{e,1} &= C_1^{(1)} \sum_{e\in E'} x''_{e,1} \\ &= C_1^{(1)}(\lfloor S_1|E'| \rfloor + 1) \\ &\leq C_1^{(1)}(S_1|E'| + 0.5). \end{aligned}$$

Similarly, for $k = 2$,

$$\begin{aligned} \sum_{e\in E'} w_{e,2}^{(1)} x''_{e,2} &= C_2^{(1)} \sum_{e\in E'} x''_{e,2} \\ &= C_2^{(1)}(\lfloor S_2|E'| \rfloor) \\ &\leq C_2^{(1)} S_2|E'|. \end{aligned}$$

Hence,

$$\begin{aligned} Z_1(P_2, x'') &= \max(\sum_{e\in E'} w^{(1)}_{e,1} x''_{e,1}, \sum_{e\in E'} w^{(1)}_{e,2} x''_{e,2}) \\ &= \max(C^{(1)}_1(S_1|E'| + 0.5), C^{(1)}_2 S_2|E'|) \\ &\leq |E'| \max(C^{(1)}_1 S_1, C^{(1)}_2 S_2) + 0.5C^{(1)}_1 \\ &= |E'| Z_1(\Pi_2(2), S) + 0.5C^{(1)}_1. \end{aligned}$$

When $dec(|E'|S_2) \geq 0.5$ it can be shown similarly that $Z_1(P_2, x'') \leq |E'|Z_1(\Pi_2(2), S) + 0.5C^{(1)}_2$. Hence, in all cases,

$$Z_1(P_2, x'') \leq |E'| Z_1(\Pi_2(2), S) + 0.5 \max_k(C^{(1)}_k).$$

Similarly, it can be shown for constraint (3.1e) that

$$Z_2(P_2, x'') \geq |E'| Z_2(\Pi_2(2), S) - 0.5 \max_k(C^{(2)}_k).$$

Step 6: Let x'' be a solution to P_2. We show there exists a solution x' for $P_1(G')$ with the objective values

- $Z_1(P_1(G'), x') = Z_1(P_2, x'')$,
- $Z_2(P_1(G'), x') = Z_2(P_2, x'')$, and
- $Z_3(P_1(G'), x') = 2$.

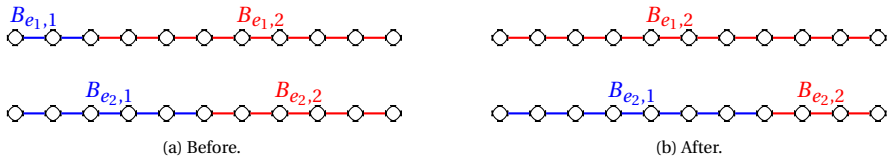
Note that $Z_3(P_1(G'), x') = 2$ means that x' partitions G' into two connected subsets.
Note that if we just simply say that $x'_{e,k} := x''_{e,k}$, we would have a solution with the above Z_1 and Z_2 values. However, the graph would look completely chaotic. This is because we have not taken the graph structure into account so far. There is however a way to transform x'' to a solution x' on graph G' resulting in only two zones.

Let A_k denote the number of edges $e \in E$ assigned to agent k in solution x''. Graph G' is 2-vertex-connected. According to Lemma 3, we can now partition the graph G' into two connected subsets of size A_1 and A_2. This partition does not change the objective values Z_1 and Z_2 and results in an objective value Z_3 equal to 2.

Step 7: Let x' be a solution to $P_1(G')$. It remains to be shown that there now exists a solution x to $P_1(G)$ with objective values

$$\begin{aligned} Z_1(P_1(G), x) &\leq Z_1(P_1(G'), x') + 0.5 \max_{e\in E}(w_e) \max_{k\in K} C^{(1)}_k \\ Z_2(P_1(G), x) &\geq Z_2(P_1(G'), x') - 0.5 \max_{e\in E}(w_e) \max_{k\in K} C^{(2)}_k \\ Z_3(P_1(G), x) &= 2. \end{aligned}$$

For the partition x' on G', we define $B_{e,k} = \{e^* \in F_e : x'_{e^*,k} = 1\}$. We call an edge $e \in E$ a demi-edge for partition x', when $|B_{e,k}| \geq 1$ for all $k \in K$. We will show that there exists a simple algorithm to change the partition x' into a partition x such that there is only one demi-edge for x and the objective values does not change.
Suppose there are two demi-edges e_1, e_2. Without loss of generality assume that $|B_{e_1,1}| \leq \min(|B_{e_1,2}|, |B_{e_2,1}|, |B_{e_2,2}|)$. We then assign all edges in $B_{e_1,1}$ to agent 2, and $|B_{e_1,1}|$ edges in $B_{e_2,2}$ to agent 1. We have now swapped $|B_{e_1,1}|$ identical edges between two agents. Hence, objectives Z_1 and Z_2 do not change. Furthermore, the new partition still results in two connected subsets. Hence $Z_3 = 2$. Edge e_1 is no longer a demi-edge. Hence, we showed that any pair of demi-edges can be reduced to a single demi-edge. The process of reducing two demi-edges to one demi-edge is shown in Figure 4.6.



(a) Before. (b) After.

Figure 4.6: Two demi-edges reduced to one demi-edge.

We can repeat this process until there is only one demi-edge e^* left. We assign this edge to $k^* := \arg\max_k(|B_{e^*,k}|)$. Doing this increases Z_1 at most with

$$0.5\, w_{e^*} C^{(1)}_{k^*} \leq 0.5 \max_{e\in E}(w_e) \max_{k\in K} C^{(1)}_k.$$

Also, Z_2 decreased with at most

$$0.5\, w_{e^*} C^{(2)}_{k^*} \leq 0.5 \max_{e\in E}(w_e) \max_{k\in K} C^{(2)}_k.$$

Z_3 remains unchanged.

Step 8:
In this step we will conclude that, for every $\delta > 0$, there exists an $n \in \mathbb{N}$ such that for all G with $|E| > n$, we have $|Z_{1,2} - Z^{OPT}_{1,2}| < \delta$.

First we bound the value of $Z^{OPT}_{1,2}$. Recall that $\Pi_2(2)$ is a relaxation of P_2, which is a relaxation of $P_1(G')$, which is a relaxation of $P_1(G)$. Hence,

$$Z^{OPT}_{1,2}(P_1(G)) \leq Z^{OPT}_{1,2}(P_1(G')) \leq Z^{OPT}_{1,2}(P_2) \leq Z^{OPT}_{1,2}(\Pi_2(2)) = \min_k \frac{C^{(1)}_k}{C^{(2)}_k}. \tag{4.14}$$

We will now bound $Z_{1,2}$. In Step 4, we showed that for S on the Pareto Front of $\Pi_2(2)$, we have

$$\frac{Z_2(\Pi_2(2), S)}{Z_1(\Pi_2(2), S)} = \min_k \frac{C^{(1)}_k}{C^{(2)}_k}. \tag{4.15}$$

In Step 5, we showed that with a solution S to $\Pi_2(2)$, we can find a solution x'' to P_2 such that

$$Z_1(P_2, x'') \leq Z_1(\Pi_2(2), S)|E'| + 0.5 \max_k(C^{(1)}_k), \tag{4.16}$$

$$Z_2(P_2, x'') \geq Z_2(\Pi_2(2), S)|E'| - 0.5 \max_k(C^{(2)}_k). \tag{4.17}$$

In Step 6, we showed that with a solution x'' to P_2, we can find a solution x' to $P_1(G')$ such that

$$Z_1(P_1(G'), x') = Z_1(P_2, x''), \tag{4.18}$$

$$Z_2(P_1(G'), x') = Z_2(P_2, x''). \tag{4.19}$$

In Step 7 we showed that with a solution x' to $P_1(G')$, we can find a solution x to $P_1(G)$ such that

$$Z_1(P_1(G), x) \leq Z_1(P_1(G'), x') + 0.5 \max_{e\in E}(w_e) \max_{k\in K} C^{(1)}_k \tag{4.20}$$

$$Z_2(P_1(G), x) \geq Z_2(P_1(G'), x') - 0.5 \max_{e\in E}(w_e) \max_{k\in K} C^{(2)}_k \tag{4.21}$$

$$Z_3(P_1(G), x) = 2 \tag{4.22}$$

We can combine Expression (4.15)- (4.22) and conclude the theorem as follows.

There exists a feasible solution x to $P_1(G)$ with $Z_3(P_1(G),x)=2$ and

$$\begin{aligned} Z_{1,2} &= \frac{Z_2(P_1(G),x)}{Z_1(P_1(G),x)} \\ &\geq \frac{Z_2(P_1(G'),x') - 0.5\max_{e\in E}(w_e)\max_{k\in K} C_k^{(2)}}{Z_1(P_1(G'),x') + 0.5\max_{e\in E}(w_e)\max_{k\in K} C_k^{(1)}} \\ &= \frac{Z_2(P_2,x'') - 0.5\max_{e\in E}(w_e)\max_{k\in K} C_k^{(2)}}{Z_1(P_2,x'') + 0.5\max_{e\in E}(w_e)\max_{k\in K} C_k^{(1)}} \\ &\geq \frac{Z_2(\Pi_2(2),S) - \frac{0.5}{|E'|}[\max_{e\in E}(w_e)+1]\max_{k\in K} C_k^{(1)}}{Z_1(\Pi_2(2),S) + \frac{0.5}{|E'|}[\max_{e\in E}(w_e)+1]\max_{k\in K} C_k^{(2)}} \\ &= \frac{Z_2(\Pi_2(2),S) - \alpha_2}{Z_1(\Pi_2(2),S) + \alpha_1} \\ &= \delta_1 \frac{Z_2(\Pi_2(2),S)}{Z_1(\Pi_2(2),S)} - \delta_2 \\ &= \delta_1 \min_k \frac{C_k^{(1)}}{C_k^{(2)}} - \delta_2. \end{aligned} \tag{4.23}$$

Here, for $i=1,2$ the $\alpha_i := \frac{0.5}{|E'|}[\max_{e\in E}(w_e)+1]\max_{k\in K} C_k^{(i)}$ go to zero when $|E|\to\infty$, since $|E'| = \sum_{e\in E} w_e$. Moreover, $\delta_1 := \frac{1}{1+\frac{\alpha_1}{Z_1(\Pi_2(2),S)}}$ goes to 1 and $\delta_2 := \frac{\alpha_2}{Z_1(\Pi_2(2),S)+\alpha_1}$ to zero as $|E|$ goes to infinity. Hence, for every δ, there exists an n such that

$$|Z_{1,2} - Z_{1,2}^{OPT}| = |\delta_1 \min_k \frac{C_k^{(1)}}{C_k^{(2)}} - \delta_2 - \min_k \frac{C_k^{(1)}}{C_k^{(2)}}| = |(\delta_1 - 1)\min_k \frac{C_k^{(1)}}{C_k^{(2)}} - \delta_2| < \delta.$$

Moreover, we have that

$$\delta_1 \min_k \frac{C_k^{(1)}}{C_k^{(2)}} - \delta_2 \leq Z_{1,2} \leq Z_{1,2}^{opt} \leq \min_k \frac{C_k^{(1)}}{C_k^{(2)}}.$$

Hence, $Z_{1,2}^{OPT} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}$.

□

Theorem 3. *We consider $P_1(G)$ with the following restrictions:*

G is a $\min(4,r)$-vertex-connected graph.
There are r agents, i.e. $K=\{1,\dots,r\}$.
$w_{e,k}^{(i)} := C_k^{(i)}$ for $i=1,2$, $k=1,\dots,r$, $e\in E$, where $C_k^{(i)}\in\mathbb{R}_+$.
$g(G,x) :=$ the number of zones in the partition x on graph G.

Then, there exists a feasible solution x with

- $Z_3 = r$.
- *For every $\delta>0$, there exists an $n\in\mathbb{N}$ such that $P_1(G)$ on a graph with $|E|=n$ has a solution with $|Z_{1,2}^{OPT} - Z_{1,2}| < \delta$. Moreover, $Z_{1,2}^{OPT} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}$.*

Proof. The proof consists of the following steps

1. We know that we can compute all solutions S on the Pareto Front of $\Pi_2(r)$.
2. Using S, we compute a solution x'' to P_2.
3. Using x'', we compute a solution x to $P_1(G)$.
4. We summarize and conclude.

Step 1: Proceeding similarly as in Step 1-3 from Theorem 2, we have that $\Pi_2(r)$ is a relaxation of $P_1(G)$. We use Lemma 1 and proceed as in Step 4 of Theorem 2 to conclude that all solutions S on the Pareto Front of $\Pi_2(r)$ have

$$\frac{Z_2(\Pi_2(r),S)}{Z_1(\Pi_2(r),S)} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}$$

Step 2: Using S, we show that we can obtain a solution x'' to P_2 with

- $Z_1(P_2,x'') \leq |E| Z_1(\Pi_2(r),S) + \max_k C_k^{(1)}$,
- $Z_2(P_2,x'') \geq |E| Z_2(\Pi_2(r),S) - \max_k C_k^{(2)}$.

To obtain x'', we assign the first S_1% of the edges to agent 1. The following S_2% of the edges, we assign to agent 2. We continue until we assigned the last S_r% of the edges to agent r. The edges that are on the "border" between agent k and $k+1$ are assigned to agent k. The process is graphically depicted for $r=4$ in Figure 4.7. It will now be formally defined.

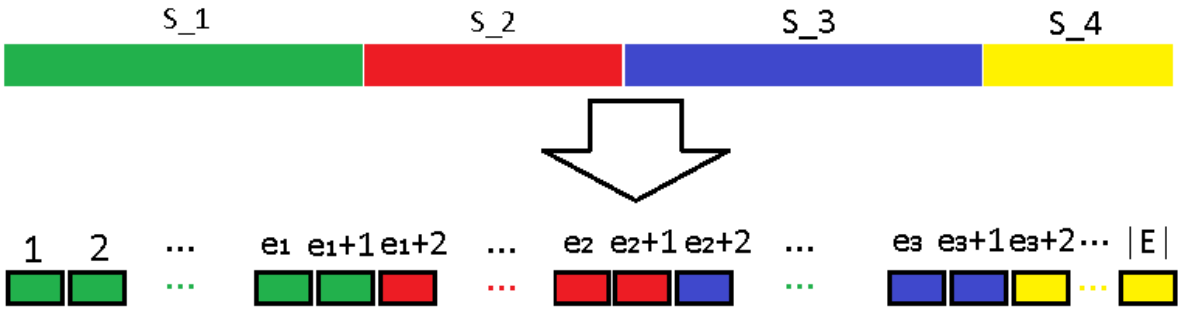


Figure 4.7: Going from solution $S = [S_1, S_2, S_3, S_4]$ to solution x''.

Let $e_0 := -1$, $e_r := |E| - 1$. For $k = 1, \ldots, r-1$, we define $e_k := \lfloor |E| \sum_{j=1}^{k} S_j \rfloor$

$$x''_{e,k} = \begin{cases} 1 & \text{if } e \in \{e_{k-1}+2, e_{k-1}+3, \ldots, e_k+1\}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that every edge gets assigned to an agent in this way (i.e. $\sum_k x''_{e,k} = 1$). Moreover, we have for constraint (3.1d) for general k

$$\begin{aligned} \sum_{e\in E} w_{e,k}^{(1)} x''_{e,k} &= C_k^{(1)} \sum_{e\in E} x''_{e,k} \\ &= C_k^{(1)} [e_k + 1 - (e_{k-1}+2) + 1] \\ &= C_k^{(1)} [e_k - e_{k-1}] \\ &\leq C_k^{(1)} [|E| \sum_{j=1}^{k} S_j - |E| \sum_{j=1}^{k-1} S_j + 1] \\ &= |E| C_k^{(1)} S_k + C_k^{(1)}. \end{aligned}$$

Therefore, we have for the objective value Z_1 of problem P_2 with solution x'' that

$$\begin{aligned} Z_1(P_2,x'') &= \max_k \sum_{e\in E} w_{e,k}^{(1)} x''_{e,k} \\ &= \max_k (|E| C_k^{(1)} S_k + C_k^{(1)}) \\ &\leq |E| \max_k (C_k^{(1)} S_k) + \max_k C_k^{(1)} \\ &= |E| Z_1(\Pi_2(r),S) + \max_k C_k^{(1)}. \end{aligned}$$

Similarly, we obtain for Z_2 that

$$Z_2(P_2, x'') \geq |E| Z_2(\Pi_2(r), S) - \max_k C_k^{(2)}.$$

Step 3:
Let x'' be a solution to P_2. We show that there exists a solution x for $P_1(G)$ with the objective values

- $Z_1(P_1(G), x) = Z_1(P_2, x'')$,
- $Z_2(P_1(G), x) = Z_2(P_2, x'')$, and
- $Z_3(P_1(G), x) = r$.

This step is very similar to Step 6 of Theorem 2. Note that $Z_3(P_1(G), x) = r$ means that x partitions G' in r zones.
Note that if we just simply say that $x_{e,k} := x''_{e,k}$, we would have a solution with the above Z_1 and Z_2 values. However, the graph would look completely chaotic. This is because we have not taken the graph structure into account so far. There is, however, a way to transform solution x'' to a solution x with only r zones in graph G.

Let A_k denote the number of edges $e \in E$ assigned to agent k in solution x''. Graph G is $\min(4, r)$-vertex-connected. According to Lemma 3, we can now partition the graph G into r connected subsets of size $A_1, A_2, \ldots, A_r$. This partition does not change the objective values Z_1 and Z_2 and results in an objective value Z_3 equal to r.

Step 4: In this step we will conclude that, for every $\delta > 0$, there exists an $n \in \mathbb{N}$ such that for all G with $|E| > n$, we have $|Z_{1,2} - Z_{1,2}^{OPT}| < \delta$.

First we bound the value of $Z_{1,2}^{OPT}$. Recall that $\Pi_2(r)$ is a relaxation of P_2, which is a relaxation of $P_1(G')$, which is a relaxation of $P_1(G)$. Hence,

$$Z_{1,2}^{OPT}(P_1(G)) \leq Z_{1,2}^{OPT}(P_1(G')) \leq Z_{1,2}^{OPT}(P_2) \leq Z_{1,2}^{OPT}(\Pi_2(r)) = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}. \tag{4.24}$$

We will now bound $Z_{1,2}$. In Step 2, we showed that for S on the Pareto Front of $\Pi_2(r)$, we have

$$\frac{Z_2(\Pi_2(r), S)}{Z_1(\Pi_2(r), S)} = \min_k \frac{C_k^{(1)}}{C_k^{(2)}}. \tag{4.25}$$

According to Step 2 we can use S to find a solution x'' to P_2 with

$$Z_1(P_2, x'') \leq |E| Z_1(\Pi_2(r), S) + \max_k C_k^{(1)}, \tag{4.26}$$

$$Z_2(P_2, x'') \geq |E| Z_2(\Pi_2(r), S) - \max_k C_k^{(2)}. \tag{4.27}$$

According to Step 3 we can use x'' to find a solution x to $P_1(G)$ with

$$Z_1(P_1(G), x) = Z_1(P_2, x''), \tag{4.28}$$

$$Z_2(P_1(G), x) = Z_2(P_2, x''), \tag{4.29}$$

$$Z_3(P_1(G), x) = r. \tag{4.30}$$

Combining Expressions (4.25)-(4.30), we obtain

$$\begin{aligned}\frac{Z_2(P_1(G),x)}{Z_1(P_1(G),x)} &= \frac{Z_2(P_2,x'')}{Z_1(P_2,x'')}\\ &\geq \frac{Z_2(\Pi_2(r),S)-\frac{1}{|E|}\max_k C_k^{(2)}}{Z_1(\Pi_2(r),S)+\frac{1}{|E|}\max_k C_k^{(1)}}\\ &= \frac{Z_2(\Pi_2(r),S)-\alpha_2}{Z_1(\Pi_2(r),S)+\alpha_1}\\ &= \delta_1\frac{Z_2(\Pi_2(r),S)}{Z_1(\Pi_2(r),S)}-\delta_2\\ &= \delta_1\min_k\frac{C_k^{(1)}}{C_k^{(2)}}-\delta_2.\end{aligned}$$

Here the $\alpha_i := \frac{1}{|E|}\max_k C_k^{(i)}$ go to zero as $|E|\to\infty$. Hence, $\delta_1 := \frac{1}{1+\frac{\alpha_1}{Z_1(\Pi_2(r),S)}}$ goes to 1 as $|E|\to\infty$ and $\delta_2 := \frac{\alpha_2}{Z_1(\Pi_2(r),S)+\alpha_1}$ goes to zero as $|E|\to\infty$.

Hence, for every δ, there exists an n such that

$$|Z_{1,2}-Z_{1,2}^{OPT}| = |\delta_1\min_k\frac{C_k^{(1)}}{C_k^{(2)}}-\delta_2-\min_k\frac{C_k^{(1)}}{C_k^{(2)}}| = |(\delta_1-1)\min_k\frac{C_k^{(1)}}{C_k^{(2)}}-\delta_2|<\delta.$$

Moreover, we have that

$$\delta_1\min_k\frac{C_k^{(1)}}{C_k^{(2)}}-\delta_2 \leq Z_{1,2}\leq Z_{1,2}^{opt}\leq \min_k\frac{C_k^{(1)}}{C_k^{(2)}}.$$

Hence, $Z_{1,2}^{OPT}=\min_k\frac{C_k^{(1)}}{C_k^{(2)}}$.

□

The value of Theorem 3 might seem limited, because we restricted the $w_{e,k}^{(i)}$ to be constants depending only on k and i. However, Theorem 3 serves as a basis for the more general case discussed in Theorem 4, where we take the $w_{e,k}^{i}$ to be identically independently distributed variables from distributions χ_k with means μ_k. In Theorem 4 we show that in this case we can still find a solution x whose value $Z_{1,2}$ gets arbitrarily close to $Z_{1,2}^{OPT}$. To prove Theorem 4 we use Lemma 2.

Theorem 4. *We consider $P_1(G)$ with the following restrictions:*

G is a $\min(4,r)$-vertex-connected graph.
There are r agents, i.e. $K=\{1,\dots,r\}$.
For all $k=1,\dots,r$, $e\in E$, $i=1,2$ the $w_{e,k}^{(i)}$ are independently identically distributed variables from distributions $\chi_k^{(i)}$ with mean $\mu_k^{(i)}$.
$g(G,x) :=$ the number of zones in the partition x on graph G.

Then, there exists a feasible solution x with

- $Z_3 = r$.
- *For every $\delta>0$, there exists an $n\in\mathbb{N}$ such that $P_1(G)$ on a graph with $|E|=n$ has a solution with $|Z_{1,2}^{OPT}-Z_{1,2}|<\delta$. Moreover, $Z_{1,2}^{OPT}=\min_k\frac{\mu_k^{(1)}}{\mu_k^{(2)}}$.*

Proof. The proof consists of the following steps

1. We know that we can compute all solutions S on the Pareto Front of $\Pi_3(r)$.
2. Using S, we compute a solution x'' to P_2.
3. Using x'', we compute a solution x to $P_1(G)$.
4. We summarize and conclude.

Step 1: We start with $\Pi_3(r)$, which is a relaxation of $P_1(G)$ that we obtain similarly as in Step 3 of Theorem 2. We use Lemma 1 and proceed as in Step 4 of Theorem 2 to conclude that all solutions S on the Pareto Front of $\Pi_3(r)$ satisfy

$$\frac{Z_2(\Pi_3(r),S)}{Z_1(\Pi_3(r),S)} = \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}}$$

Step 2: We will show that with a solution S to $\Pi_3(r)$ we can find a solution x'' to P_2 that satisfies

- $Z_1(P_2,x'') \leq |E| Z_1(\Pi_3(r),S) + A^{(1)}$,
- $Z_2(P_2,x'') \geq |E| Z_2(\Pi_3(r),S) + A^{(2)}$,

where $A^{(1)} := \max_{k\in K}\left\{|E|S_k|Q_k|^{-0.5+\epsilon} + |Q_k|^{-0.5+\epsilon} + \mu_k^{(1)}\right\}$ and $A^{(2)} := \min_{k\in K}\left\{-|E|S_k|Q_k|^{-0.5+\epsilon} - \mu_k^{(2)} + |Q_k|^{-0.5+\epsilon}\right\}$.

We find x'' using S in the exact same manner as in Step 2 of Theorem 3: Let $e_0 := -1$, $e_r := |E|-1$. For $k = 1,\ldots,r-1$, we define $e_k := \lfloor |E| \sum_{j=1}^{k} S_j \rfloor$. Let $Q_k := \{e_{k-1}+2, e_{k-1}+3, \ldots, e_k+1\}$. We now take $x''_{e,k}$ to be

$$x''_{e,k} = \begin{cases} 1 & \text{if } e \in Q_k, \\ 0 & \text{otherwise.} \end{cases}$$

We now have for constraint (3.1d) for general k:

$$\begin{aligned}
\sum_{e\in E} w_{e,k}^{(1)} x''_{e,k} &= \sum_{e\in Q_k} w_{e,k}^{(1)} \\
&= |Q_k| \frac{\sum_{e\in Q_k} w_{e,k}^{(1)}}{|Q_k|} \\
&= |Q_k| \left(\frac{\sum_{e\in Q_k} w_{e,k}^{(1)}}{|Q_k|} - \mu_k^{(1)} \right) + |Q_k| \mu_k^{(1)}. \qquad (4.31)
\end{aligned}$$

Note that when $|E| \to \infty$ then $|Q_k| \to \infty$ too. Let $0 < \epsilon < 0.5$. When we let $|E|$ go to infinity, we can use Lemma 2 to bound the parts between brackets in Expression (4.31) to obtain

$$\begin{aligned}
\sum_{e\in E} w_{e,k}^{(1)} x''_{e,k} &\leq |Q_k||Q_k|^{-0.5+\epsilon} + |Q_k|\mu_k^{(1)} \\
&= |Q_k| \left(|Q_k|^{-0.5+\epsilon} + \mu_k^{(1)} \right) \\
&\leq (|E|S_k + 1) \left(|Q_k|^{-0.5+\epsilon} + \mu_k^{(1)} \right) \\
&= |E|S_k\mu_k^{(1)} + |E|S_k|Q_k|^{-0.5+\epsilon} + |Q_k|^{-0.5+\epsilon} + \mu_k^{(1)}.
\end{aligned}$$

We can now bound $Z_1(P_2,x'')$ from above

$$\begin{aligned}
Z_1(P_2,x'') &= \max_k \left(\sum_{e\in E} w_{e,k}^{(1)} x''_{e,k} \right) \\
&\leq |E| \max_k \left[S_k(\mu_k^{(1)}) \right] + A^{(1)} \\
&= |E| Z_1(\Pi_3(r),S) + A^{(1)}.
\end{aligned}$$

For constraint (3.1e) and general k, we proceed similarly, to obtain:

$$Z_2(P_2,x'') \geq |E| Z_2(\Pi_3(r),s) + A^{(2)}.$$

Step 3: Let x'' be a solution to P_2. Proceeding as in Step 3 of Theorem 3, there exists a solution x for $P_1(G)$ with the objective values

- $Z_1(P_1(G),x) = Z_1(P_2,x'')$,
- $Z_2(P_1(G),x) = Z_2(P_2,x'')$, and
- $Z_3(P_1(G),x) = r$.

Step 4: In this step we will conclude that, for every $\delta > 0$, there exists an $n \in \mathbb{N}$ such that for all G with $|E| > n$, we have $|Z_{1,2} - Z_{1,2}^{OPT}| < \delta$.

First we bound the value of $Z_{1,2}^{OPT}$. Recall that $\Pi_3(r)$ is a relaxation of $P_1(G)$. Hence,

$$Z_{1,2}^{OPT}(P_1(G)) \leq Z_{1,2}^{OPT}(\Pi_2(r)) = \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}}. \tag{4.32}$$

We will now bound $Z_{1,2}$. According to Step 1, we can find solutions S to $\Pi_3(r)$ with

$$\frac{Z_2(\Pi_3(r),S)}{Z_1(\Pi_3(r),S)} = \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}}. \tag{4.33}$$

According to Step 2 we can use S to find a solution x'' to P_2 with

$$Z_1(P_2,x'') \leq |E| Z_1(\Pi_3(r),S) + A^{(1)}, \tag{4.34}$$

$$Z_2(P_2,x'') \geq |E| Z_2(\Pi_3(r),S) + A^{(2)}. \tag{4.35}$$

According to Step 3 we can use x'' to find a solution x to $P_1(G)$ with

$$Z_1(P_1(G),x) = Z_1(P_2,x''), \tag{4.36}$$

$$Z_2(P_1(G),x) = Z_2(P_2,x''), \tag{4.37}$$

$$Z_3(P_1(G),x) = r. \tag{4.38}$$

Combining Expression (4.33)-(4.38), we have

$$\begin{aligned}
\frac{Z_2(P_1(G),x)}{Z_1(P_1(G),x)} &= \frac{Z_2(P_2,x'')}{Z_1(P_2,x'')} \\
&\geq \frac{Z_2(\Pi_3(r),S) - \frac{1}{|E|}A^{(1)}}{Z_1(\Pi_3(r),S) + \frac{1}{|E|}A^{(2)}} \\
&= \delta_1 \frac{Z_2(\Pi_3(r),S)}{Z_1(\Pi_3(r),S)} - \delta_2 \\
&= \delta_1 \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}} - \delta_2.
\end{aligned}$$

Here, $\frac{A^{(1)}}{|E|}$ and $\frac{A^{(2)}}{|E|}$ go to zero as $|E| \to \infty$. Hence, $\delta_1 := \frac{1}{1+\frac{\frac{A^{(1)}}{|E|}}{Z_1(\Pi_3(r),S)}}$ goes to 1 as $|E| \to \infty$ and $\delta_2 := \frac{\frac{A^{(2)}}{|E|}}{Z_1(\Pi_3(r),S)+\frac{A^{(1)}}{|E|}}$ goes to zero as $|E| \to \infty$.
Hence, for every δ, there exists an n such that

$$|Z_{1,2} - Z_{1,2}^{OPT}| = |\delta_1 \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}} - \delta_2 - \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}}| = |(\delta_1 - 1)\min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}} - \delta_2| < \delta.$$

Moreover, we have that

$$\delta_1 \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}} - \delta_2 \leq Z_{1,2} \leq Z_{1,2}^{opt} \leq \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}}.$$

Hence, $Z_{1,2}^{OPT} = \min_k \frac{\mu_k^{(1)}}{\mu_k^{(2)}}$.

□

We have arrived at the last theorem. In this theorem we again assume G to be a $\min(4,r)$-vertex-connected graph with r agents. In the previous theorem, we considered one distribution $\chi_k^{(i)}$ for each agent. We now consider the distributions $\chi^{(1)}$ and $\chi^{(2)}$ from which we draw the values $w_e^{(1)}$ and $w_e^{(2)}$. Each agent now has a constant C_k. This setting resembles the debris management problem well. The value w_e can be seen as the amounts of debris, different for every street. The constants C_k resembles the efficiency rate for every agent. The smaller C_k, the faster it can clean the debris.

Theorem 5. *We consider $P_1(G)$ with the following restrictions:*

G is a $\min(4,r)$-vertex-connected graph.
There are r agents, i.e. $K = \{1,\dots,r\}$.
For all $k = 1,\dots,r$, $e \in E$, $i = 1,2$, we have $w_{e,k}^{(i)} = w_e^{(i)} C_k$, where the $w_e^{(i)}$'s are identically distributed variables from a distribution $\chi^{(i)}$ with mean μ_i.
$g(G,x) :=$ the number of zones in the partition x on graph G.

Then, for every $\delta > 0$, there exists a $n \in \mathbb{N}$ such that $P_1(G)$ on a graph with $|E| = n$ has a solution x with

- $Z_3 = r$,
- $\frac{|Z_1^{OPT} - Z_1|}{Z_1^{OPT}} < \delta$,
- $\frac{|Z_2^{OPT} - Z_2|}{Z_2^{OPT}} < \delta$.

Proof. The proof consists of the following steps:

1. Compute a lower bound for Z_1 and an upper bound for Z_2 in P_2.
2. Find a solution x' to P_2 which gets arbitrarily close to these lower bounds as $|E|$ goes to infinity.
3. Using x', we compute a solution x to $P_1(G)$ that meets the requirements.

Step 1: We compute a lower bound for Z_1 and an upper bound for Z_2 in problem P_2. In problem P_2 we relax the binary constraint. Or put differently, we replace the constraint $x_{e,k} \in \{0,1\}$ with $x_{e,k} \geq 0$. It turns out that this relaxed problem has a solution x'' that minimizes Z_1 and maximizes Z_2 at the same time. This solution has

$$x''_{e,k} = \frac{1}{C_k \sum_j \frac{1}{C_j}} \qquad \forall e,k$$

and gives us the following bounds for Z_1 and Z_2 in P_2 for any feasible solution x

$$Z_1(P_2,x) \geq \frac{\sum_e w_e^{(1)}}{\sum_j \frac{1}{C_j}}, \tag{4.39}$$

$$Z_2(P_2,x) \leq \frac{\sum_e w_e^{(2)}}{\sum_j \frac{1}{C_j}}. \tag{4.40}$$

We define these lower bounds respectively as $Z_1^{(low)}$ and $Z_2^{(high)}$.

Step 2: We find a solution x' to P_2 that gets arbitrarily close to these lower bounds as $|E|$ goes to infinity.

We create x' in a similar manner as in Step 2 of Theorem 3. Let $l_k = \frac{1}{C_k \sum_j \frac{1}{C_j}}$. Now let $e_0 := -1$, $e_r := |E| - 1$. For $k = 1,\dots,r-1$, we define $e_k := \lfloor |E| \sum_{j=1}^{k} l_j \rfloor$. Let $Q_k := \{e_{k-1}+2, e_{k-1}+3, \dots, e_k+1\}$. We now take $x'_{e,k}$ to be

$$x'_{e,k} = \begin{cases} 1 & \text{if } e \in Q_k, \\ 0 & \text{otherwise.} \end{cases}$$

Let k' be the agent for which $Z_1(P_2,x) = \sum_{e\in E} w_e^{(1)} C_{k'} x'_{e,k'}$

$$\begin{aligned}
\left(Z_1(P_2,x') - Z_1^{(low)}\right) &= \left(\sum_{e\in E} w_e^{(1)} C_{k'} x'_{e,k'} - \frac{\sum_e w_e^{(1)}}{\sum_j \frac{1}{C_j}}\right) \\
&= \left(C_{k'} \sum_{e\in Q_{k'}} w_e^{(1)} - \frac{\sum_e w_e^{(1)}}{\sum_j \frac{1}{C_j}}\right) \\
&= \frac{1}{\sum_j \frac{1}{C_j}} \left(\sum_j \frac{1}{C_j} C_{k'} \sum_{e\in Q_{k'}} w_e^{(1)} - \sum_{e\in E} w_e^{(1)}\right) \\
&= \frac{1}{\sum_j \frac{1}{C_j}} \left(\frac{|E|}{|Q_{k'}|} \sum_{e\in Q_{k'}} w_e^{(1)} - \sum_{e\in E} w_e^{(1)}\right) \\
&\leq \frac{1}{\sum_j \frac{1}{C_j}} t|E|^{0.5},
\end{aligned}$$

where t is a constant. In the last step we used Lemma 2. We now have

$$\frac{\left(Z_1(P_2,x') - Z_1^{(low)}\right)}{Z_1^{(low)}} = o(|E|^{-0.5}). \tag{4.41}$$

Hence, as $|E|$ increases $Z_1(P_2,x')$ gets arbitrarily close to its lower bound. Similarly, it can be shown that, as $|E|$ increases $Z_2(P_2,x')$ gets arbitrarily close to its upper bound.

Step 3: Let x' be a solution to P_2. Proceeding as in Step 3 of Theorem 3, there exists a solution x for $P_1(G)$ with the objective values

- $Z_1(P_1(G),x) = Z_1(P_2,x')$,
- $Z_2(P_1(G),x) = Z_2(P_2,x')$, and
- $Z_3(P_1(G),x) = r$.

This gives the result. □

4.2. The multi-resource assignment problem

In Chapter 2 we introduced the Multi-Resource Generalized Assignment Problem, or MRGAP. In this section we give the Mixed Integer Program (MIP) of this problem and give an algorithm that solves the problem when we allow to violate two constraints slightly. The relevance of our algorithm for this thesis is not straightforward, since graphs are not mentioned in it. However, in the next chapter, we will see that the results obtained in this section are, in fact, applicable to the Unrelated Unconnected Multi-Constrained Graph Partitioning problem (UUMCGP) studied in this thesis.

We denote the MIP of the MRGAP by P_{MRGAP}. It is defined as follows:

$$P_{MRGAP} = \begin{cases}
\min \quad C := \sum_{e,k} c_{e,k} x_{e,k} & & (4.42a) \\
\text{s.t.} \quad \sum_{e\in E} w_{e,k}^{(1)} x_{e,k} \leq Z_1 & \forall k \in K & (4.42b) \\
\sum_{e\in E} w_{e,k}^{(2)} x_{e,k} \leq Z_2 & \forall k \in K & (4.42c) \\
\sum_{k\in K} x_{e,k} = 1 & \forall e \in E & (4.42d) \\
x_{e,k} \in \{0,1\} & \forall e \in E, k \in K & (4.42e)
\end{cases}$$

Our algorithm resembles a bi-criteria algorithm. Iyer and Bilmes [19] define this as an approximation algorithm that violates one constraint. In our case there is not one, but two violated constraints. We call this class of algorithms tri-criteria algorithms. We define them as follows.

Definition 3. *Let Q be a set of constraints and let P be a program defined by*

$$P = \min\{f(x) | \quad g(x) \leq \phi_1, h(x) \leq \phi_2, Q\},$$

where x is a solution to P, f is the function to be minimized and $g(x)$ and $h(x)$ are constraints. Let P^{OPT} be the value of $f(x)$ at a feasible and optimal x.
Now, for $\alpha, \beta, \gamma \geq 1$, $\mathbb{A}$ is an (α, β, γ)-tri-criteria algorithm when it outputs a solution x that has

$f(x) \leq \alpha P^{opt}$,
$g(x) \leq \beta\phi_1$, *and*
$h(x) \leq \gamma\phi_2$.

In other words, $\mathbb{A}$ gives a solution x that gives an objective value a factor α close to its optimal, while violating the first constraint with a factor β and the second constraint with a factor γ.

Our algorithm is an extension of a bi-criteria algorithm for the Generalized Assignment Problem (GAP) developed by Lenstra et al.[22]. We denote the MIP of GAP by P_{GAP}. It is defined as P_{MRGAP} but without constraint (4.42c). Lenstra et al. [22] showed that for P_{GAP} a bi-criteria algorithm exists that shows that either no feasible solution exists, or outputs a solution x at optimal C, while only violating constraint (4.42b) by $w^{(1)}_{max} := \max_{e,k} w^{(1)}_{e,k}$. We will now extend this algorithm so it can be applied to P_{MRGAP}. We first need two definitions.

Definition 4. *Two vectors $(a_1, \ldots, a_m), (b_1, \ldots, b_m) \in \mathbb{R}^m$ are δ-ordered if*

- $\frac{a_{i+1}}{a_i} \leq \delta$ *for all* $i \in [1, \ldots, m-1]$ *and*
- $\frac{b_{i+1}}{b_i} \leq \delta$ *for all* $i \in [1, \ldots, m-1]$.

Note that for all $a, b \in \mathbb{R}^m$, a and b are δ'-ordered, with $\delta' := \max(\frac{a_{max}}{a_{min}}, \frac{b_{max}}{b_{min}})$, where a_{max} and b_{max} are the maximum entries of a and b, and a_{min} and b_{min} are the minimum entries of a and b. Moreover, note that when we have $a_1 \geq \cdots \geq a_m$ and $b_1 \geq \cdots \geq b_m$, the vectors are $\delta'-$ordered with $\delta' \leq 1$.

We can almost start proving the existence of the algorithm. We only need one more definition. We will define what we mean by a fractional complete matching. We will use this term in our proof.

Definition 5. *Let $G = (N, M, F)$ be a bipartite graph, where F is the set of edges connecting the node set N with the node set M. Then y is a fractional complete matching of G, if and only if all of the following hold.*

$$\sum_{m:(n,m)\in F} y_{n,m} = 1 \text{ for all } n \in N, \tag{4.43}$$

$$\sum_{n:(n,m)\in F} y_{n,m} \leq 1 \text{ for all } m \in M, \tag{4.44}$$

$$y_{n,m} \geq 0 \text{ for all } n \in N \text{ and } m \in M. \tag{4.45}$$

Moreover, y is an integer complete matching of G, when all of the above hold, but instead of Expression (4.45) *we have $y_{n,m} = \{0, 1\}$ for all $n \in N$ and $m \in M$.*
Moreover, the cost of a matching y is $\sum_{n,m} c_{n,m} y_{n,m}$, where $c_{n,m} \in \mathbb{R}_{\geq 0}$ for all $n \in N$, $m \in M$.

We can now state and prove the theorem.

Theorem 6. *Consider P_{MRGAP}. Assume that for all $k \in K$ the vectors $(w^{(1)}_{1,k}, \cdots, w^{(1)}_{m,k})$ and $(w^{(2)}_{1,k}, \cdots, w^{(2)}_{m,k})$ are δ-ordered. Then there exist a $(1, 1 + \max(1, \delta)^{|E|}, 1 + \max(1, \delta)^{|E|})$-tri-criteria algorithm to MRGAP*

Proof.
We will output a solution x in polynomial time that has the following.

- $\sum_{e,k} c_{e,k} x_{e,k} \leq C_{opt}$,
- $\sum_{e \in E} w^{(1)}_{e,k} x_{e,k} \leq \epsilon^{|E|} Z_1 + w^{(1)}_{max}$,
- $\sum_{e \in E} w^{(2)}_{e,k} x_{e,k} \leq \epsilon^{|E|} Z_2 + w^{(2)}_{max}$,

where C_{opt} is the value of C in the optimal solution, $\epsilon := \max(1, \delta)$ and $w^{(i)}_{max} := \max_{e,k} w^{(i)}_{e,k}$ for $i = 1,2$. Since $w^{(i)}_{max} \leq Z_i$ for $i = 1,2$, the theorem then follows.

This proof resembles a proof concerning the P_{GAP} given in [35] and consists of the following steps:

1. We find the optimal solution $x'_{e,k}$ to a relaxation of P_{MGRAP} with costs $C' \leq C_{opt}$.
2. We use $x'_{e,k}$ to construct a bipartite graph B.
3. We show that B has a fractional complete matching of cost at most C'.
4. We show that B has an integer complete matching of costs at most C'.
5. We show that every integer complete matching of B of costs C' corresponds to a solution to P_{MRGAP} of costs at most C_{opt}.
6. We show that every integer complete matching of B corresponds to a solution to P_{MRGAP} in which each agent requires at most $\epsilon^{|E|} Z_1 + w^{(1)}_{max}$ of the first resource and $\epsilon^{|E|} Z_2 + w^{(2)}_{max}$ of the second resource.

Step 1: We can relax P_{MRGAP} by replacing constraint (4.42e) by

$$x_{e,k} \geq 0 \quad \forall e \in E, k \in K.$$

The new problem can be solved in polynomial time. Let x' denote the optimal solution to this problem. Note that x' is a fractional solution. Let C' denote the cost of this solution. Hence, $C' := \sum_{e,k} c_{e,k} x'_{e,k}$. Since x' is the optimal solution to the relaxation of P_{MRGAP}, we have that $C' \leq C_{opt}$.

We can depict x' as a bipartite graph A with node sets E and K. We connect edge e with agent k if $x'_{e,k} > 0$. An example of a bipartite graph A corresponding to a solution x' is given in Figure 4.8a.

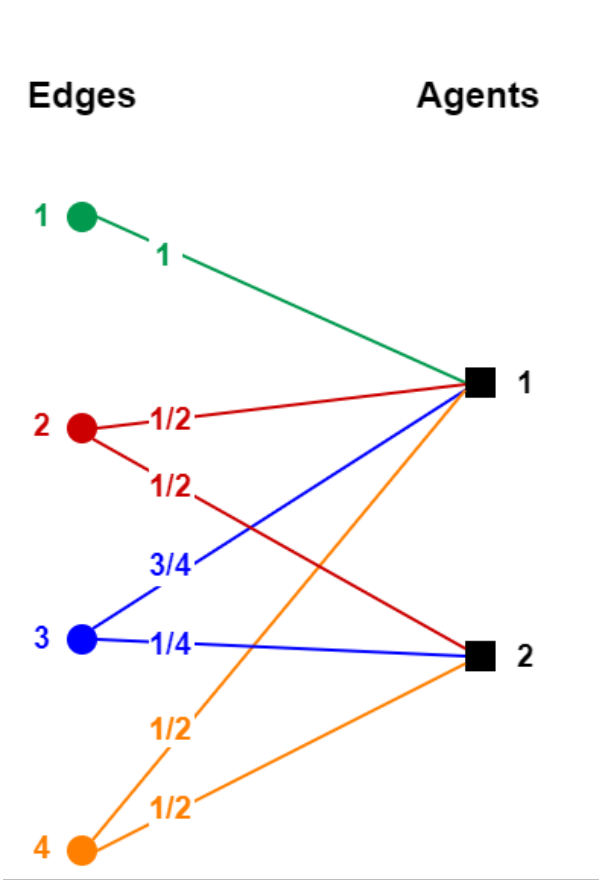
Step 2: We will now construct a new bipartite graph B using x'. Later on, we will use this graph to prove our results. Let $L_k := \lceil \sum_{e \in E} x_{e,k} \rceil$, where $\lceil a \rceil$ denotes the integer obtained from rounding up the number a to the nearest integer. Now, for every agent k we define "agent slots" (k,s) for $s = 1,2,\ldots,L_k$. Let S denote the set containing all agent slots of all agents. $S := \{(k,s), \quad k = 1,2,\ldots,r \text{ and } s = 1,2,\ldots,L_k\}$. The two node sets of our bipartite graph are now E and S. To determine the edges F of our bipartite graph B we use a so-called "bucket filling" procedure.

This procedure works as follows. For every agent k we start connecting nodes with slot $(k,1)$. We first select the node e_1 with $x'_{e_1,k} > 0$ with the lowest index. We connect e_1 to $(k,1)$ and we fill slot $(k,1)$ with $x'_{e_1,k}$. We then select the node e_2 with $x'_{e_2,k} > 0$ with the second lowest index. We again connect this node e_2 with slot $(k,1)$ and add $x'_{e_2,k}$ to the slot. We repeat this process until the slot is completely filled and we continue filling the next slot $(k,2)$. When $(k,2)$ is filled up, we start filling $(k,3)$. We continue until all nodes e for which $x'_{e,k} > 0$ are connected to slots (k,s). During this procedure we define a variable y. Whenever we pack a positive fraction of e in slot (k,s), we set $y_{e,(k,s)}$ equal to that fraction. This procedure is depicted in Figure 4.8b.

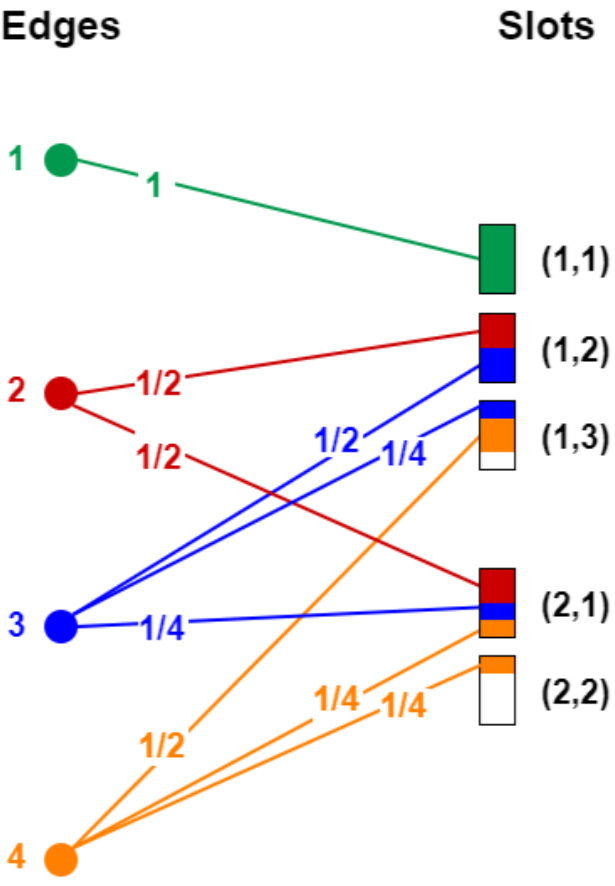
Step 3: Note that the variable y is a fractional complete matching in B. When we define the cost function $c_{e,(k,s)}$ to be $c_{e,k}$, we have that the cost of the fractional complete matching y is equal to C', since

$$\begin{aligned}
\sum_e \sum_k \sum_s c_{e,(k,s)} y_{e,(k,s)} &= \sum_e \sum_k \sum_s c_{e,k} y_{e,(k,s)} \\
&= \sum_e \sum_k c_{e,k} \sum_s y_{e,(k,s)} \\
&= \sum_e \sum_k c_{e,k} x'_{e,k} \\
&= C'.
\end{aligned}$$

Step 4: Every bipartite graph that has a fractional complete matching contains an integer complete matching with smaller or equal costs [35]. Hence, there exists an integer complete matching z of graph B with costs



(a) Bipartite graph A. The value on every edge (e,k) denotes $x'_{e,k}$.



(b) Bipartite graph B. The value on every edge $(e,(k,s))$ denotes $y_{e,(k,s)}$.

Figure 4.8: An example of the bipartite graphs A and B corresponding to a solution x'. In this example there are 4 edges that need to be assigned to 2 agents.

smaller or equal than C'.

Step 5: Recall that z is the integer complete matching of graph B with costs smaller or equal than C'. Hence, $\sum_e \sum_k \sum_s c_{e,(k,s)} z_{e,(k,s)} \leq C'$. We will show now that there exists an integer solution x to P_{MRGAP} that might be infeasible, but that has costs at most C_{opt}. We define this x as follows:

$$x_{e,k} = \begin{cases} 1 & \text{if } z_{e,(k,s)} = 1 \text{ for some } s = 1,2,\ldots,L_k, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $x_{e,k}$ is integer. Note that $x_{e,k} = \sum_s z_{e,(k,s)}$. Therefore we have that

$$\begin{aligned} \sum_e \sum_k c_{e,k} x_{e,k} &= \sum_e \sum_k c_{e,k} \sum_s z_{e,(k,s)} \\ &= \sum_e \sum_k \sum_s c_{e,k} z_{e,(k,s)} \\ &\leq C' \\ &\leq C_{opt}. \end{aligned}$$

Step 6: In Step 5 we found an integer solution $x_{e,k}$ that has costs at most C_{opt}. It remains to show that this $x_{e,k}$ violates the constraints only by the required amount. We will show this only for constraint (4.42b). The proof for constraint (4.42c) is the same.

We will show that for any agent k, the total load of the first weight assigned to him will be lower than the required amount $\epsilon^{|E|} Z_1 + w^{(1)}_{max}$. We know that the vector $(w^{(1)}_{1,k}, \cdots, w^{(1)}_{m,k})$ is δ-ordered. Hence, we have

$$w^{(1)}_{m,k} \leq \delta w^{(1)}_{m-1,k} \leq \delta^2 w^{(1)}_{m-2,k} \leq \ldots \leq \delta^{m-1} w^{(1)}_{1,k}.$$

When we set $\epsilon = \max(1, \delta)$, we get

$$w^{(1)}_{m,k} \leq \epsilon w^{(1)}_{m-1,k} \leq \epsilon^2 w^{(1)}_{m-2,k} \leq \ldots \leq \epsilon^{m-1} w^{(1)}_{1,k}.$$

We therefore have for every $e \in E$

$$w^{(1)}_{e,k} \leq \epsilon^{|E|} w^{(1)}_{e',k} \quad \text{for all } e' < e \tag{4.46}$$

In Step 4 we gave the integer complete matching z on the bipartite graph B. Note that the total load for agent k under the solution x, is the same as the total load for agent k under the integer complete matching z. It therefore suffices to show that agent k has a load smaller than $\epsilon^{|E|} Z_1 + w^{(1)}_{max}$ in the integer complete matching z.

Note that graph B only contained edges $(e,(k,s))$ for which $y_{e,(k,s)} > 0$. Hence the only edges in z that can be connected to an agent slot (k,s) are $(e,(k,s))$ for e such that $y_{e,(k,s)} > 0$. Let $max(k,s)$ denote the maximum value of $w^{(1)}_{e,k}$ among those edges. Hence, the total load of agent k in the integer matching z is not larger than

$$\begin{aligned}\sum_{s=1}^{L_k} max(k,s) &= max(k,1) + \sum_{s=2}^{L_k} max(k,s) \\ &\leq w^{(1)}_{max} + \sum_{s=2}^{L_k} max(k,s)\end{aligned}$$

It remains to show that $\sum_{s=2}^{L_k} max(k,s) \leq \epsilon^{|E|} Z_1$. Because we assigned the edges with higher indices first in the bucket filling procedure, we have that all the e' assigned to an agent slot (k,s) have a lower index than the edges assigned to agent slot $(k,s+1)$. Hence, we use (4.46) to obtain $max(k,s+1) \leq \epsilon^{|E|} w^{(1)}_{e,k}$ for all e connected to agent slot (k,s). Now also note that for $s = 1,\ldots,L_k - 1$ the agents slots are filled, i.e. $\sum_e y_{e,(k,s)} = 1$. Hence, for $s = 1,\ldots,L_k - 1$,

$$max(k,s+1) \leq \sum_e y_{e,(k,s)} \epsilon^{|E|} w^{(1)}_{e,k}.$$

Hence, we get

$$\begin{aligned}\sum_{s=2}^{L_k} max(k,s) &= \sum_{s=1}^{L_k-1} max(k,s+1) \\ &\leq \sum_{s=1}^{L_k-1} \sum_e y_{e,(k,s)} \epsilon^{|E|} w^{(1)}_{e,k} \\ &\leq \sum_{s=1}^{L_k} \sum_e y_{e,(k,s)} \epsilon^{|E|} w^{(1)}_{e,k} \\ &= \epsilon^{|E|} \sum_e w^{(1)}_{e,k} \sum_{s=1}^{L_k} y_{e,(k,s)}\end{aligned}$$

But, by construction of y, we have $\sum_{s=1}^{L_k} y_{e,(k,s)} = x'_{e,k}$, where $x'_{e,k}$ is the optimal fractional solution to P_{MRGAP} from Step 1. Hence,

$$\begin{aligned}\sum_{s=2}^{L_k} max(k,s) &\leq \epsilon^{|E|} \sum_e w^{(1)}_{e,k} x'_{e,k} \\ &\leq \epsilon^{|E|} Z_1\end{aligned}$$

□

Note that this theorem is strong in particular when the relevant vectors are δ'-ordered with $\delta' \leq 1$. In that case we can find an optimal solution to $MRGAP$ in polynomial time while only violating Z_i by $w^{(i)}_{max}$ for $i = 1,2$. In the next chapter we will use Theorem 6 to come up with an algorithm to approximately solve $P_1(G)$.

5
Algorithms

The multi-objectivity of $P_1(G)$ makes it hard or impossible to solve it in practice. Moreover, Theorem 1 in Section 4.1.1 shows that $P_1(G)$ is an NP-hard problem even when it is single objective in Z_1 or Z_2. This motivates the use of approximation algorithms to find good solutions. In this chapter such an algorithm is presented. We take the function $g(G,x)$ in $P_1(G)$ as the sum of the spans of all nodes as defined in Expression (3.2).

In Section 4.2 we defined the multi-resource generalized assignment problem, or MRGAP. The mixed integer problem (MIP) of MRGAP is denoted by P_{MRGAP} and is given in (4.42). In this problem we assign edges to agents so that the cost function is minimized. Moreover, every agent can use at most Z_1 of the first resource and Z_2 of the second resource. We have seen that there exists a tri-criteria algorithm for P_{MRGAP}. The algorithms discussed in this section are based on this result, since they repeatedly solve the P_{MRGAP}. We call these algorithms "UM" algorithms. Here the "U" stands for the problem the algorithm approximately solves: the Unconnected Unrelated Multi-Constrained Graph Partitioning problem. The "M" stand for the MIP it uses to come up with this solution: the Multi-Resource Generalized Assignment Problem. We distinguish two types of these UM algorithms: "starting" UM algorithms and "ending" UM algorithms. First, in Section 5.1 we will explain the general workings of the basic UM algorithm. In Section 5.2 we will then dive into the starting UM algorithms. The ending UM algorithms will be explored in Section 5.3. In Section 5.4 we will conclude and present the final algorithm.

5.1. The UM algorithm

UM algorithms use a certain edge-agent assignment y with good objective values Z_1 and Z_2 as input. They then solve the multi-resource assignment problem P_{MRGAP} to obtain an edge-agent assignment x with an improved Z_3 objective, while only slightly worsening the Z_1 and Z_2 objectives. It then uses the new edge-agent assignment as an input and repeats the process. We will now present the extra notation of this section and the exact model and steps of the UM algorithm.

Parameters:

$$y_{e,k} = \begin{cases} 1 & \text{if edge } e \text{ is assigned to agent } k \text{ in the } \textit{input} \text{ assignment,} \\ 0 & \text{otherwise.} \end{cases}$$

$h_{e,k}(y)$ Parameter depending on given assignment y.

B_1 Parameter denoting the upper bound of Z_1.

B_2 Parameter denoting the lower bound of Z_2.

Decision variables:

$$x_{e,k} = \begin{cases} 1 & \text{if edge } e \text{ gets assigned to agent } k \text{ in the new assignment,} \\ 0 & \text{otherwise.} \end{cases}$$

Model:

Suppose we have a graph $G = (V,E)$ with agent-depended edge weights $w^{(1)}_{e,k}$ and $w^{(2)}_{e,k}$. Let $y_{e,k}$ be a given edge-agent assignment. We can now write P_{UM}, which is a mixed-integer program (MIP) very similar to P_{MRGAP}.

Recall that P_{MRGAP} does not take any graph into account. To include the graph structure of problem $P_1(G)$, we replace in P_{MRGAP} the parameter $c_{e,k}$ with a new parameter $h_{e,k}(y)$ that we calculate based on characteristics of the graph and the assignment y. More on that calculation in the next sections. We also replace Z_1 with B_1, Z_2 with B_2 and flip the inequality sign in (4.42c). We then obtain P_{UM}.

$$P_{UM} = \begin{cases} \min Z_{UM} := \sum_{e,k} h_{e,k}(y) x_{e,k} & \text{(5.1a)} \\ s.t. \sum_{k \in K} x_{e,k} = 1 \quad \forall e & \text{(5.1b)} \\ \sum_{e \in E} w^{(1)}_{e,k} x_{e,k} \leq B_1 \quad \forall k & \text{(5.1c)} \\ \sum_{e \in E} w^{(2)}_{e,k} x_{e,k} \geq B_2 \quad \forall k & \text{(5.1d)} \\ x_{e,k} \in \{0,1\} \quad \forall k, e & \text{(5.1e)} \end{cases}$$

Constraint (5.1b) makes sure that every edge gets assigned to exactly one agent. Constraint (5.1c) makes sure that the first objective Z_1 is bounded from above by B_1. Constraint (5.1d) makes sure that the second objective Z_2 is bounded from below by B_1. Hence, the solution x is an edge assignment with objective values Z_1 and Z_2 bounded by respectively B_1 and B_2. There is one more thing left to show: x has a better chaos measure than the old edge assignment y. This improvement is due to the choice of $h_{e,k}(y)$ in the objective value Z_{UM}. We will elaborate on this choice in the following sections. The objective value Z_{UM} can be seen as the increase in chaos between the input assignment y and the new assignment x. A negative objective value of Z_{UM} corresponds to a decrease in chaos between the assignments.

In the following we will describe two classes of the UM algorithm: starting and ending UM algorithms. To solve an instance of $P_1(G)$ one first runs a starting UM algorithm and ends with an ending UM algorithm. Each class, starting and ending, has 3 versions: the regular, restricted, and combination version. The versions only differ in their $h_{e,k}(y)$ parameter. By changing this parameter each version reduces the chaos of the old assignment in its own way.

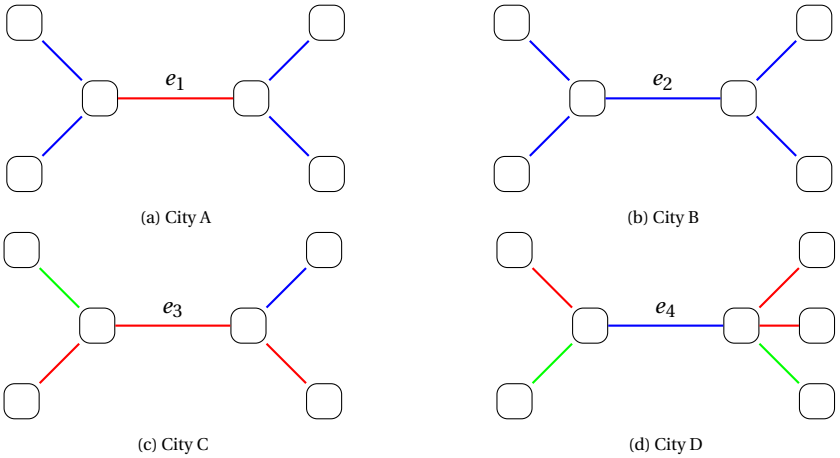
5.2. Starting UM algorithms

Imagine a solution y to $P_1(G)$ with good values of Z_1 and Z_2 but with a bad objective value Z_3. To improve the latter, one can run one of the three versions of the starting UM algorithm. We will discuss each version now and conclude which version is the best starting UM algorithm.

5.2.1. The regular version

The algorithm

We define the parameter $h(y)$ calculated in the regular version by $h^{(1)}(y)$. We define $h^{(1)}_{e,k}(y)$ as the increase in the chaos objective, defined in Expression (3.2), when changing edge e to agent k in assignment y. Some examples of edges with their corresponding $h^{(1)}_{e,k}$ values are given in Figure 5.1 and Table 5.1.



(a) City A (b) City B

(c) City C (d) City D

Figure 5.1: Four different edge-agent assignments in example cities with 3 agents (red, blue and green).

Edge	red	blue	green
e_1	0	-2	0
e_2	2	0	2
e_3	0	1	1
e_4	-2	0	-2

Table 5.1: $h^{(1)}_{e,k}$ values corresponding to edges in Figure 5.1.

Notice that for an edge e that is already assigned to agent k in the old assignment y (i.e. when $y_{e,k} = 1$), we have $h^{(1)}_{e,k}(y) = 0$. Note that, when we do not change the solution (i.e. $x_{e,k} = y_{e,k}$), the objective value Z_{UM} in Expression 5.1a equals zero.
Let all parameters be given as in $P_1(G)$ and P_{UM}. Like the other versions, the regular version of the UM algorithm takes an assignment y with good Z_1 and Z_2 objective values. It then solves P_{UM} obtaining a solution x. It repeats this process, each time using the obtained solution x as an input for the new iteration, until the objective value Z_3 does not improve anymore. The algorithm is shown in Algorithm 1.

Algorithm 1 The regular version of the starting UM algorithm

1: Find solution y of $P_1(G)$ with good Z_1, Z_2 values
2: $\Delta := -1$
3: **while** $\Delta < 0$ **do**
4: $\quad h_{e,k}(y) \leftarrow h^{(1)}_{e,k}(y) \forall e, k$
5: $\quad$ Find optimal solution x to P_{UM}
6: $\quad \Delta \leftarrow Z_{UM}(x)$
7: $\quad y \leftarrow x$

The Results
The results after five iterations of the regular version of the UM algorithm are shown in Figures 5.2 and 5.3.

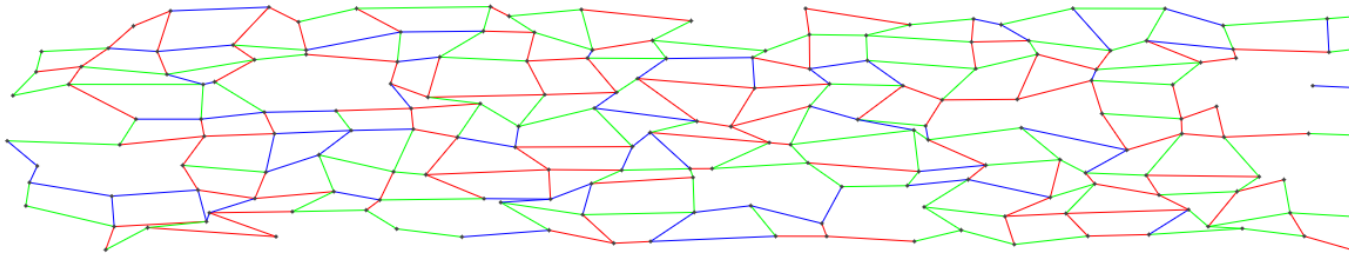


Figure 5.2: Edge-agent assignment.

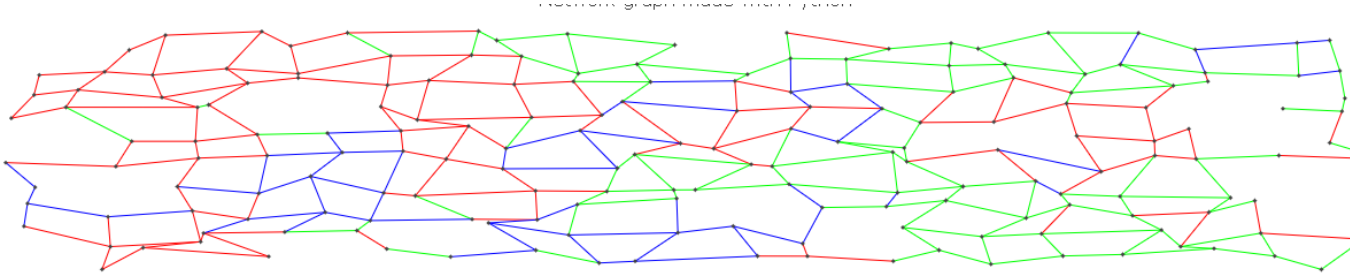


Figure 5.3: Edge-agent assignment after five iterations of the regular version of the UM algorithm.

It can be seen that the chaos of the graph has been decreased after five iterations. However, parts of the graph are still chaotically assigned. Running more iterations does not solve this. This is due to the adjacent-edge-problem, which is described below.

Adjacent-edge-problem
The main problem with the regular version of the UM algorithm is that it does not take into account that adjacent edges can both change agent. An example of this is given in Figures 5.4 and 5.5.

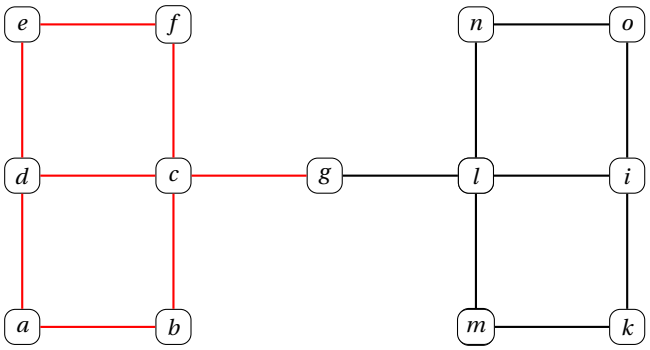


Figure 5.4: Edge-agents assignment for a simple city with two agents.

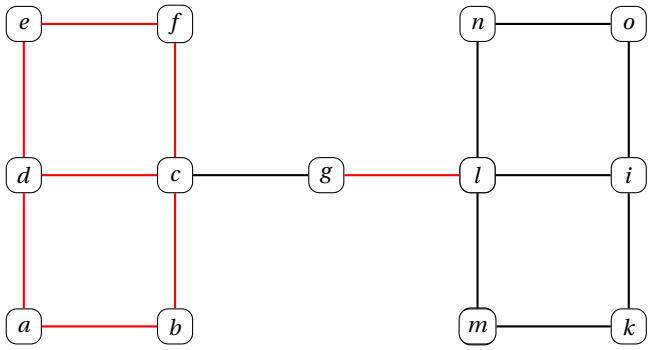


Figure 5.5: Edge-agents assignment of simple city with two agents after iteration of running the regular version of the UM algorithm

In Figure 5.4, swapping the edge (c,g) to the black agent does not worsen the sum of spans defined in Expression (3.2), nor does swapping the edge (g,l) to the red agent. Running the UM algorithm on this instance, therefore might result in the edge-agent assignment depicted in 5.5. The objective value will be zero, but the chaos of the assignment has increased. This problem will be dealt with in the restricted version of the UM algorithm.

5.2.2. The restricted version

The algorithm

We define the parameter $h(y)$ calculated in the restricted version by $h^{(2)}(y)$. In the restricted version we solve the adjacent-edge-problem by making sure that for every node only one edge adjacent to that node can change agent. We do this using a matching. A matching in a graph $G = (V,E)$ is a subset E' of the edges such that no edges in E' are adjacent to the same node. In short, in the restricted version, we first construct a matching as described below. We then assign values to $h^{(2)}_{e,k}$ for all e and k such that all edges in the matching can change agent and all edges not in the matching stay with their current agent. Since a matching does not contain adjacent edges, we make sure that there are no adjacent edges that can both change agent.

We want to have the edges in the matching, which, when changed to another agent, lead to the biggest reduction in chaos. We search for a minimum cost matching on G with the edge weights for each edge e equal to $\min_k\{h^{(1)}_{e,k}(y)\}$. The problem of finding a minimum cost matching can be solved in time $O(|V|^2|E|)$ using Edmonds' blossom algorithm [6]. For simplification reasons, we use a Greedy algorithm to find a feasible matching. In each step we take, from a set of available edges, the edge with the highest potential for decreasing the chaos. Or, put differently, we search for the edge e with the smallest value of $\min_k\{h^{(1)}_{e,k}(y)\}$. We add this edge to the matching, and remove this edge, together with all adjacent edges from the set of available edges. We repeat until there are no more available edges. The process is shown in Algorithm 2.

Algorithm 2 Find a minimum cost matching M

1: $A := E$
2: $M := \emptyset$
3: **while** $A \neq \emptyset$ **do**
4: $\quad e^* \leftarrow \arg\min_{e\in A}[\min_k h^{(1)}_{e,k}(y)]$
5: $\quad A \leftarrow A \setminus \{e^*\}$
6: $\quad A \leftarrow A \setminus \{f\}$ for all $f \in A$ adjacent to e^* in G
7: $\quad M \leftarrow M \cup \{e^*\}$

Now that we have a matching M, the values of $h^{(2)}_{e,k}$ can be assigned. All edges in the matching M get assigned the same $h_{e,k}$ as in the regular version of the UM algorithm. The edges not in M get assigned $h_{e,k}$ values in such a way that they are forced to stay with their current agent. Hence, we define $h^{(2)}_{e,k}$ by

$$h^{(2)}_{e,k}(y) := \begin{cases} h^{(1)}_{e,k}(y) & \text{if } e \in M, \\ 0 & \text{if } e \notin M \text{ and } y_{e,k} = 1, \\ \infty & \text{if } e \notin M \text{ and } y_{e,k} = 0. \end{cases}$$

In our code we use 10^9 instead of ∞. The adjacent-edge problem is solved by this way of assigning values to $h_{e,k}$. We can look at Figure 5.4 to see this. Let us say that the edge (c,g) is in the matching M. Then edge (g,l) is not. The values for $h^{(1)}_{e,k}(y)$ and $h^{(2)}_{e,k}(y)$ for Figure 5.4 are shown in Table 5.2

Edge e	**Agent k**	$h^{(1)}_{e,k}$	$h^{(2)}_{e,k}$
(c,g)	**red**	0	0
(c,g)	**black**	0	0
(g,l)	**red**	0	∞
(g,l)	**black**	0	0

Table 5.2: Values for $h^{(1)}_{e,k}(y)$ and $h^{(2)}_{e,k}(y)$ in Figure 5.4.

When we solve P_{UM} with the values $h_{e,k} := h^{(2)}_{e,k}$ the new solution x will assign edge (g,l) to the black agent and thereby avoiding the situation depicted in Figure 5.5. The restricted version of the starting UM algorithm is shown in Algorithm 3.

Algorithm 3 The restricted version of the UM algorithm

Find solution y of $P_1(G)$ with good Z_1, Z_2 values
$\Delta := -1$
while $\Delta < 0$ **do**
    Find matching M with algorithm 2

    for $e \in M$ **do**
        $h^{(2)}_{e,k}(y) \leftarrow h^{(1)}_{e,k}(y) \quad \forall k$

    for $e \notin M$ **do**
        if $y_{e,k} = 1$ **then**
            $h^{(2)}_{e,k}(y) \leftarrow 0$
        if $y_{e,k} = 0$ **then**
            $h^{(2)}_{e,k}(y) \leftarrow \infty$

    $h_{e,k}(y) \leftarrow h^{(2)}_{e,k}(y) \quad \forall e,k$
    Find optimal solution x to P_{UM}
    $\Delta \leftarrow Z_{UM}(x)$
    $y \leftarrow x$

The disadvantage of solving P_{UM} with $h_{e,k} := h^{(2)}_{e,k}$ is that only around 25% of the edges are allowed to change agent, since only edges in the matching are allowed to switch. Therefore, the results of the restricted version are similar to the those of the regular version. To solve this problem we developed the last version of the starting UM algorithms: the combination version.

5.2.3. The combination version

So far, we have discussed two versions of the starting UM algorithm: the regular and the restricted version. Both give decent results, but are constrained for different reasons. The performance of the regular version is held back by the adjacent-edge problem and the performance of the restricted version is held back due to the fact that only around 25% of the edges are allowed to change agent. To solve this we run a hybrid form of these two versions, which we call the combination version. This version repeatedly alternates between the regular and restricted version. Its exact workings are shown in Algorithm 4.

Algorithm 4 The combination version of the starting UM algorithm

Find solution y of $P_1(G)$ with good Z_1, Z_2 values
$\Delta := -1$
while $\Delta < 0$ **do**
    $h_{e,k}(y) \leftarrow h^{(1)}_{e,k}(y)$
    Find optimal solution x to P_{UM}
    $\Delta_1 \leftarrow Z_{UM}(x)$
    $y \leftarrow x$

    $h_{e,k}(y) \leftarrow h^{(2)}_{e,k}(y)$
    Find optimal solution x to P_{UM}
    $\Delta_2 \leftarrow Z_{UM}(x)$
    $y \leftarrow x$

    $\Delta = \Delta_1 + \Delta_2$

Results
In Figure 5.6 a chaotic graph was transformed by the combination version of the UM algorithm to a “nicer” graph while only hurting the time and profit objective by 5%.

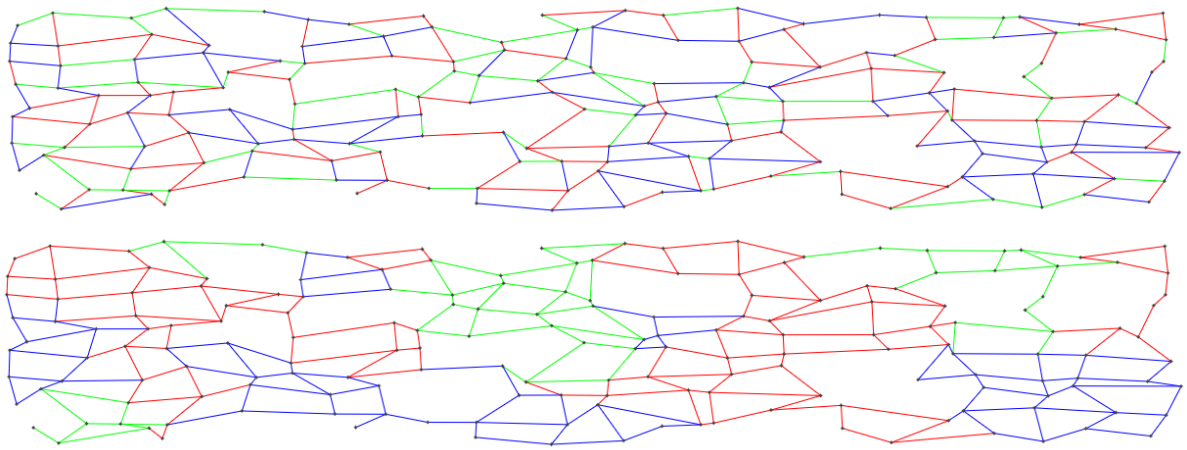


Figure 5.6: Edge-agent assignment before and after running the combination algorithm.

Looking at Figure 5.6 gives the impression that the combination version gives better results than the regular or restricted version. This impression is confirmed in the following section.

5.2.4. Conclusion

We have discussed three versions of starting UM algorithms. The results of these are shown in Figure 5.7.

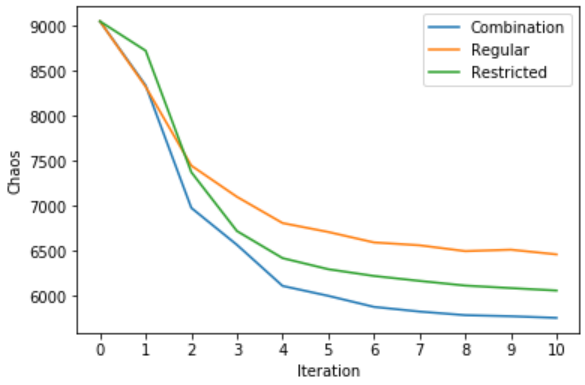


Figure 5.7: Chaos decrease measured in the sum of span for all starting versions of the UM algorithm on a simulated disaster in the city of Nijmegen, the Netherlands with 5 contractors. Nijmegen has 170.000 inhabitants and 5500 streets.

In Figure 5.7 we see, first of all, the effectiveness of the starting UM algorithms. After only 10 iterations all versions have significantly reduced the chaos. More on the effectiveness of these algorithms follows in the results chapter of this thesis. Secondly, we see that, as expected, the combination version performs better than the other algorithms. This pattern reoccurs in other experiments we ran. Thirdly, we see that the chaos-reducing effect of the algorithms wears out after about 10 iterations resulting in assignments similar to the one in Figure 5.6. The chaos has been significantly reduced. However, agents still have multiple non-connected regions of edges assigned to it. Moreover, some regions are as small as three or four edges. There is definitely more work to do. The next class of UM algorithms does this work. The ending class of UM algorithms, described in the following section, transforms the solution produced by the starting UM algorithm into an even less chaotic desirable final solution.

5.3. Ending UM algorithms

The previous class of UM algorithms produced solutions y to $P_1(G)$ with good objective values Z_1 and Z_2 and with a decent chaos objective Z_3. The class covered in this section improves the chaos objective of these solutions further. They are the last step in approximately solving $P_1(G)$ and are therefore called ending UM algorithms.

They operate very similarly to the starting UM algorithms with the difference that they operate on zones instead of on edges. Here, a zone is defined in Definition 1. Let Φ denote the set of all zones F. For every $e \in E$, let F_e denote the zone containing e. Figure 5.8 depicts the zones for a certain edge agent assignment.

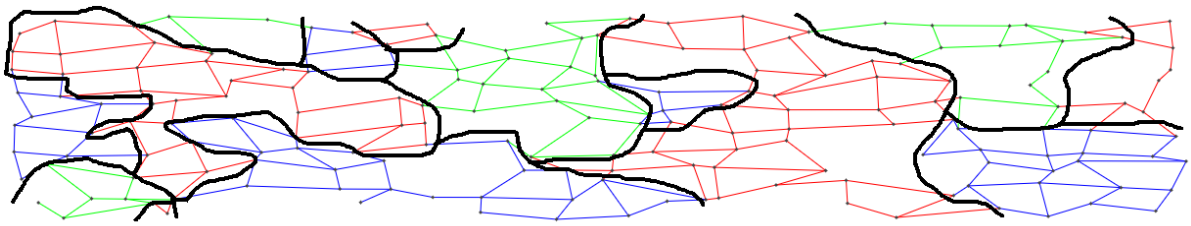


Figure 5.8: Zones for a certain edge agent assignment.

Now, instead of assigning edges to agents, this class of algorithms assigns zones to agents. We name the MIP, which is approximately solved during the ending UM algorithms, P^Z_{UM}. Approximately solving P^Z_{UM} changes an old assignment y^Z to a new assignment x^Z with reduced chaos. To obtain P^Z_{UM} from P_{UM}, we simply replace the set E of edges by the set Φ. The decision variables $x_{e,k}$ then logically become $x^Z_{F,k}$ and the parameter $h_{e,k}$ changes to $h_{F,k}$. We now replace the weight parameter $w^{(i)}_{e,k}$ by $w^{(i)}_{F,k} := \sum_{e \in F} w^{(i)}_{e,k}$. Note that a

solution x^Z to P^Z_{UM} corresponds to a solution x to P_{UM} as follows:

$$x_{e,k} = \begin{cases} 1 & \text{if } x^Z_{F_e,k} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Just like the starting UM algorithms, we distinguish three versions: the regular, restricted and combination version. All of them approximately solve P^Z_{UM} iteratively. Again, they only differ in the values of the parameter $h_{F,k}(y^Z)$

5.3.1. The regular version

We define the parameter $h(y^Z)$ calculated in the regular version of the ending UM algorithm by $h^{(3)}(y^Z)$. The values of $h^{(3)}(y^Z)$ are chosen in such a way that zones are forced to switch agent so that their new agent matches that of adjacent zones. Running the regular version repeatedly will then have small zones merging to larger zones resulting in a reduced chaos.

To formally define $h^{(3)}(y)$ we need to define $N(F)$ first. With $N(F)$ we denote the set of agents assigned to zones adjacent to F. Or, more formally:

$$N(F) = \{k \in K : y_{F',k} = 1 \text{ for some } F' \text{ adjacent to } F\}$$

We can now define $h^{(3)}(y)$:

$$h^{(3)}(y)_{F,k} = \begin{cases} 0 & \text{if } y^Z_{F,k} = 1, \\ -1 & \text{if } k \in N(F), \\ \infty & \text{otherwise.} \end{cases}$$

An example of how to determine the value $h^{(3)}(y^Z)$ is given in Figure 5.9 and Table 5.3.

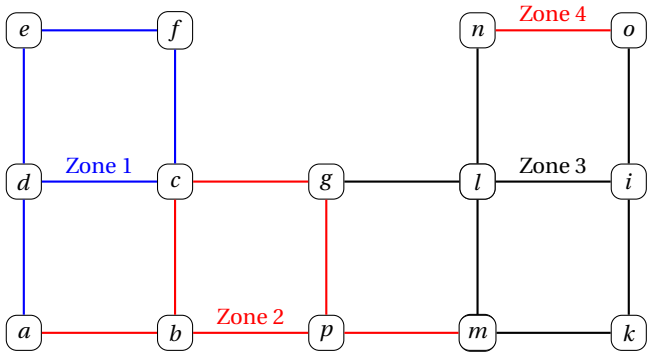


Figure 5.9: Four zones of edge-agent assignment of simple city with three agents.

		Agent		
		Blue	**Red**	**Black**
Zone	**1**	0	-1	∞
	2	-1	0	-1
	3	∞	-1	0
	4	∞	0	-1

Table 5.3: Values of $h^{(3)}_{F,k}(y^Z)$ for the edge assignment depicted in Figure 5.9

Note that, when we do not change the solution, (i.e $x^Z_{F,k} = y^Z_{F,k}$), the objective value Z_{UM} in Expression (5.1a) equals zero. The regular version of the ending UM algorithm is shown in Algorithm 5. The regular version of the ending UM algorithm gives good results. However, just like the adjacent-edge-problem in the regular version of the starting UM algorithm, it struggles with adjacent elements. In this case, adjacent zones. Adjacent zones can both switch agent leading to increased instead of reduced chaos. To deal with this problem, the restricted version of the ending UM algorithm was developed.

Algorithm 5 The regular version of the ending UM algorithm

$\Delta := -1$
$y \leftarrow$ Edge assignment obtained by running a starting UM algorithm
while $\Delta < 0$ **do**
    Obtain zone assignment y^Z from edge assignment y
    $h_{F,k}(y^Z) \leftarrow h^{(3)}_{F,k}(y^Z) \quad \forall e,k$
    Find optimal solution x^Z to P^Z_{UM}
    $\Delta \leftarrow Z_{UM}(x^Z)$
    Obtain edge assignment x from zone assignment x^Z
    $y \leftarrow x$

5.3.2. The restricted version

We define the parameter $h(y^Z)$ calculated in the regular version of the ending UM algorithm by $h^{(4)}(y^Z)$. The values of $h^{(4)}(y^Z)$ are chosen in such a way that adjacent zones can not both switch agent. We do this by selecting a subset $M \subseteq \Phi$ of zones that we allow to change. All zones not contained in M are forced to stay to their current agent. We choose M in such a way that it i) does not contain adjacent zones and ii) does contain the zones with the least amount of edges. The algorithm of finding M is similar to the matching algorithm described in Algorithm 2 and is depicted in Algorithm 6.

Algorithm 6 Find a subset M of Φ

$A := \Phi$
$M := \emptyset$
while $A \neq \emptyset$ **do**
    $F^* \leftarrow \arg\min_{F \in A}[|F|]$
    $A \leftarrow A \setminus \{F^*\}$
    $A \leftarrow A \setminus \{F'\}$ for all $F' \in A$ adjacent to F^* in G
    $M \leftarrow M \cup \{F^*\}$

When we have this subset M, we can construct our parameter $h^{(4)}(y^Z)$ as follows:

$$h^{(4)}_{F,k}(y^Z) := \begin{cases} h^{(3)}_{F,k}(y^Z) & \text{if } F \in M, \\ 0 & \text{if } F \notin M \text{ and } y^Z_{F,k} = 1, \\ \infty & \text{if } F \notin M \text{ and } y^Z_{F,k} = 0. \end{cases}$$

The algorithm of the restricted version of the ending UM algorithm is depicted in Algorithm 7.

Algorithm 7 The restricted version of the ending UM algorithm

$y \leftarrow$ Edge assignment obtained by running a starting UM algorithm
$\Delta := -1$
while $\Delta < 0$ **do**
    Obtain zone assignment y^Z from edge assignment y
    Find subset $M \subseteq \Phi$ with Algorithm 6

    for $F \in M$ **do**
        $h^{(4)}_{F,k}(y^Z) \leftarrow h^{(3)}_{F,k}(y^Z) \quad \forall k$

    for $F \notin M$ **do**
        if $y^Z_{F,k} = 1$ **then**
            $h^{(4)}_{F,k}(y^Z) \leftarrow 0$
        if $y^Z_{F,k} = 0$ **then**
            $h^{(4)}_{F,k}(y^Z) \leftarrow \infty$

    $h_{e,k}(y^Z) \leftarrow h^{(4)}_{e,k}(y^Z) \quad \forall e,k$
    Find optimal solution x^Z to P^Z_{UM}
    $\Delta \leftarrow Z_{UM}(x^Z)$
    Obtain edge assignment x from zone assignment x^Z
    $y \leftarrow x$

The restricted version deals with the problem the regular version was struggling with. However, it restrains many zones from switching agent, since it only allows adjacent agents to switch agent. Therefore, the results of the restricted version are similar then the ones of the regular version in chaos. To solve this, we came up with the combination version of the ending UM algorithm.

5.3.3. The combination version

The regular version and restricted version work well, but both have their disadvantages. Again, we need to combine the two versions to obtain the best result. This is what we do in the combination version of the ending UM algorithm. This version repeatedly alternates between the regular and restricted version. Its exact workings are shown in Algorithm 8.

Algorithm 8 The combination version of the ending UM algorithm

$y \leftarrow$ Edge assignment obtained by running a starting UM algorithm
$\Delta := -1$
while $\Delta < 0$ **do**
    Obtain zone assignment y^Z from edge assignment y
    $h_{F,k}(y^Z) \leftarrow h^{(3)}_{F,k}(y^Z)$
    Find optimal solution x^Z to P^Z_{UM}
    $\Delta_1 \leftarrow Z_{UM}(x^Z)$
    Obtain edge assignment x from zone assignment x^Z
    $y \leftarrow x$

    Obtain zone assignment y^Z from edge assignment y
    $h_{F,k}(y^Z) \leftarrow h^{(4)}_{F,k}(y^Z)$
    Find optimal solution x^Z to P^Z_{UM}
    $\Delta_2 \leftarrow Z_{UM}(x^Z)$
    Obtain edge assignment x from zone assignment x^Z
    $y \leftarrow x$

    $\Delta = \Delta_1 + \Delta_2$

5.3.4. Conclusion

We have discussed three versions of ending UM algorithms. Figures 5.10 and 5.11 show how these algorithms decrease the sum of span and the number of zones in a particular example.

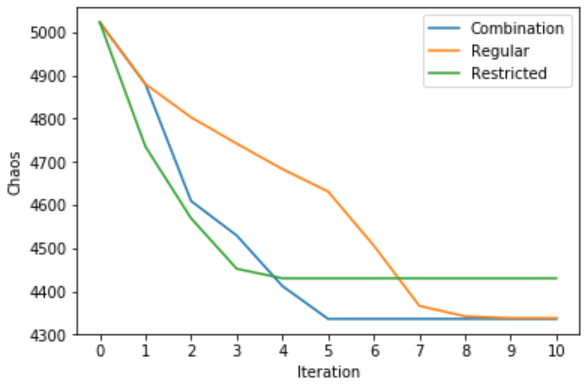


Figure 5.10: Decrease in sum of span for all ending versions of the UM algorithm on a simulated disaster in the city of Nijmegen, the Netherlands with 5 contractors. Nijmegen has 5500 streets.

From Figure 5.10 and 5.11 we can draw several conclusions. Again, just like in the case of the starting version of the UM algorithm, the algorithms are effective in reducing the chaos and the number of zones. Moreover, it can be concluded that the combination version works best: it reduces the chaos and the number of zones the most of all versions. Lastly, we see that the chaos reducing effect wears out after 10 iterations.

5.4. Conclusion

We have discussed two classes of algorithms, starting and ending. Each with a regular, restricted and combination version. We concluded that for both classes the combination version is the most effective in reducing the chaos. This conclusion is used to create the “final UM algorithm”. The final UM algorithm simply runs the combination version of the starting UM algorithm first and ends by running the combination version of the ending UM algorithm. The final UM algorithm is shown in Algorithm 9.

In Figure 5.12 and Figure 5.13 it can be seen how a chaotic graph is turned into a less chaotic graph using the “final UM algorithm”.

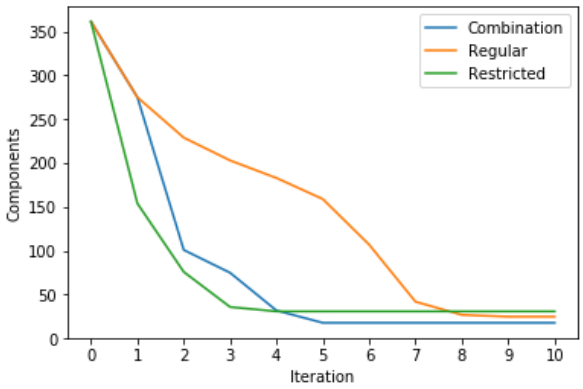
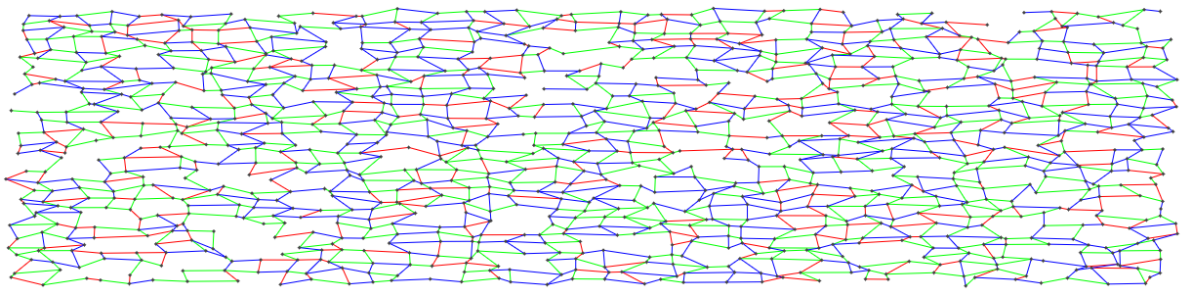


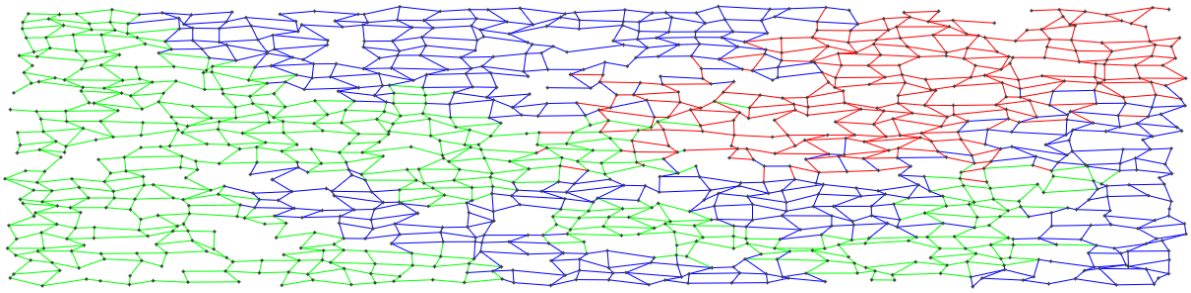
Figure 5.11: Decrease in number of zones for all ending versions of the UM algorithm on a simulated disaster in the city of Nijmegen, the Netherlands with 5 contractors. Nijmegen has 5500 streets.

Algorithm 9 The final UM algorithm

1: Run Algorithm 4 (The combination version of the starting UM algorithm)
2: Run Algorithm 8 (The combination version of the ending UM algorithm)



(a) Original chaotic graph before UM algorithm.



(b) Graph after running the final UM algorithm given in Algorithm 9.

Figure 5.12: In this 30 by 40 node graph with three agents, the chaos is successfully reduced. The objective values Z_1 and Z_2 are still around 85% of their optimal values.

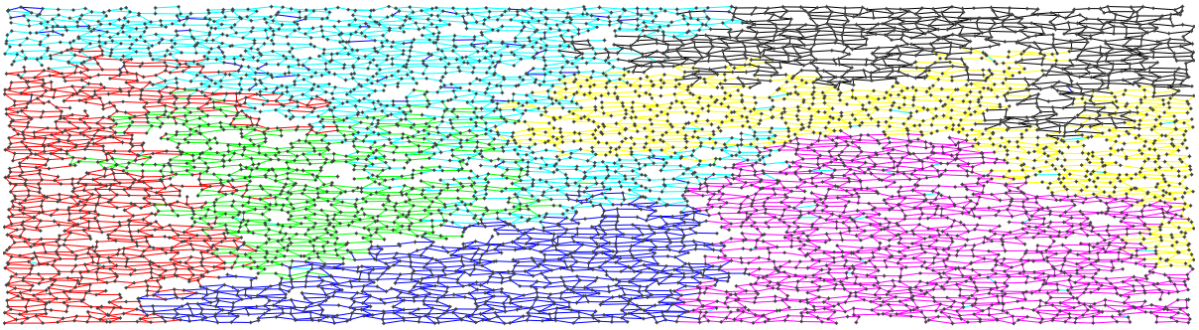


Figure 5.13: Result after running the final UM algorithm on an 80 by 80 node graph with seven agents.

Figure 5.12 and Figure 5.13 show the power of the final UM algorithm. However, we still do not know much about the quality of our algorithm. Does it perform well on all types of graphs and all types of edge weights? How good are the solutions to $P_1(G)$ it produces for all three objective values? How does it perform on real-life instances like cities? Does it outperform an MIP solver? We answer these questions by testing the algorithm on different instances. In the next chapter we will discuss how we created these instances and how we test our algorithms. We will then show the results.

6

Experimental Design

To test the performance of the final UM algorithm discussed in the previous chapter, we run computational experiments. In this chapter we discuss on which instances we test the algorithms (Section 6.1) and how we test them (Section 6.2). We will test our algorithms on real-life cities where we will simulate disasters. For this reason we will speak of time instead of Z_1, profit instead of Z_2 and contractors instead of agents.

6.1. Creating instances

There are three types of parameters that influence our problem $P_1(G)$: the number of contractors, the graph G and the weights $w_{e,k}^{(i)}$. After we make choices for each of these parameters, we obtain an instance of a city after a natural disaster.

Number of contractors

We seek to answer the question: what is the impact of the number of contractors on the performance of the final UM algorithm? We test the algorithm for 5 and 10 contractors. This captures the variation that occurs for real-life debris management problems.

Graphs

Another question we seek to answer is the following: does the city street plan have an impact on the performance of the algorithms? To answer this question we test our algorithms on a set of 9 American cities with the highest Natural Disaster Hazard Score. This is a score measuring the combined risk of earthquakes, fires, floods, tornadoes and hurricanes [8]. The nine cities are Miami (Florida), Sacramento (California), Seattle (Washington), Memphis (Tennessee), Philadelphia (Pennsylvania), St. Louis (Missouri), Birmingham (Alabama), San Antonio (Texas) and Kansas City (Missouri). Street maps of all cities were downloaded from Open Street Map using Osmnx in Python [2]. For each city we remove all self loops for simplification reasons. These are streets that start at the same point as that they end. We can do this without significantly hurting the output, because self-loops form less 1% of the streets.

The weights $w_{e,k}^{i}$

In $P_1(G)$ every edge has two weights for every contractor: $w_{e,k}^1$ and $w_{e,k}^2$. In the applied case of natural disasters the first weight $w_{e,k}^1$ represents the time it takes for an contractor k to clean street e. The second weight $w_{e,k}^2$ represents the profit it takes for contractor k to clean street e. There are many ways that we can assign values to these weights and each assignment may have an effect on the performance of the final UM algorithm. Regarding the edge weights, we hope to answer the following questions, the terminology of which will be explained below.

1. What is the impact of symmetric versus asymmetric contractors on the performance of our algorithm?
2. What is the impact of an equal spread of debris versus a non-equal spread of debris on the performance of our algorithm?
3. What is the impact of a correlated time and profit and a non-correlated time and profit?

We will now elaborate on these questions and how we will create 8 weight settings to answer them. In Table 6.2 an overview of all weights settings and their corresponding values is provided.

The first question deals with the symmetry of the contractors. We can have contractors who all have similar efficiency, machine power and number of trucks. This would make computations easier. In reality however, more often than not, the contractors differ. One can imagine for example that some contractors have more and bigger trucks, better drivers or more efficient truck engines using less fuel. We mimic asymmetry between contractors by dividing edge weights by a factor c_k for each contractor k. A contractor with a high value of c_k corresponds to an efficient contractor with many trucks. When there are r contractors, we assign the values c_k such that $[c_1, c_2, \ldots, c_r] = [1, 2, \ldots, r]$.

The second question deals with the distribution of the debris over the city. It makes computation easier when debris is more or less equally spread within the city. In reality however, some parts of cities tend to have more debris than other parts. This has two main reasons. First, more populated parts of the city create more debris and second, the intensity of the disaster is different in different parts of the city. The destruction of an earthquake is bigger closer to the epicentre, the damage of a cyclone is bigger closer to the eye of the storm and the damage of floods is closer to the sea or river. In this thesis, we only consider the first reason. In the following we will explain why the physical structure of the disaster is not taken into account as a cause of an unequal debris spread.

Let us start with earthquakes. We will argue that its intensity is more or less the same across a city. The destruction of an earthquake can be measured using the unit intensity I ranging from 1 (no damage) to 12 (total damage, destroyed bridges and buildings). The intensity I of the earthquake at a point at the surface x can be estimated using equations including the magnitude M of the earthquake and the distance R_{diag} of the point x to the hypocentre, the point in the crust where the earthquake takes place [1]. Using Pythagoras' theorem we can replace the distance R_{diag} in these equations with $\sqrt{R_d^2 + R_v^2}$, where R_d is the horizontal distance between x and the epicentre and R_v is the depth of the earthquake. R_{diag}, R_d and R_v are graphically depicted in Figure 6.1

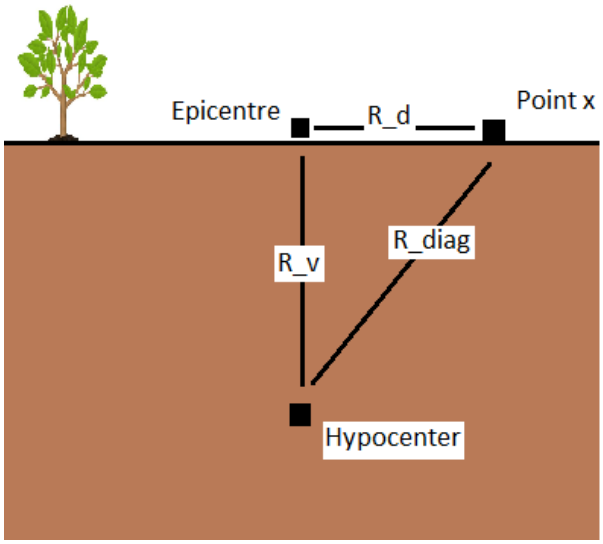
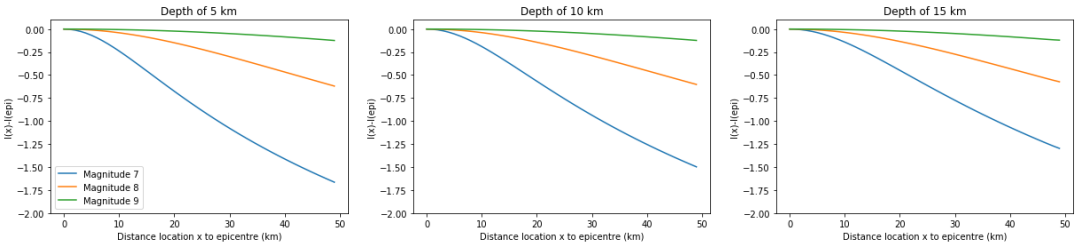


Figure 6.1: Depiction of R_{diag}, R_h and R_v

We can now plot the difference in intensity between a point x and the epicentre with the distance R_d between point x and the epicentre. We can do this for different magnitudes and depths of the earthquake. The results are shown in Figure 6.2.



(a) Depth of 5 km (b) Depth of 10 km (c) Depth of 15 km

Figure 6.2: Difference in earthquake intensities between a point x and the epicentre for different earthquake magnitudes and depths.

As expected the intensity of the earthquake decreases as we move further away from the epicentre. However, it decreases slowly. A magnitude 7 earthquake occurring at a depth of 5 km is felt 1.75 intensities lower 50 km away from the epicentre. A magnitude 9 earthquake at a depth of 15 km is felt just 0.1 intensity lower 50 km away from the epicentre. We conclude that cities are too small to have significant differences in earthquake intensity.

For hurricanes we can draw a similar conclusion. The further from the eye of the hurricane the smaller the destruction, but the size of a hurricane-force wind field averages about 161 km across. The area over which tropical storm-force winds occur is even greater, ranging as far out as almost 500 km from the eye of a large hurricane [14]. We conclude that cities are too small to have significant differences in hurricane intensity.
We just showed that the intensity of earthquakes and hurricanes can be assumed similar across a city. This is not the case however for floods, wildfires, tornadoes and landslides. These types of disasters cause more damage in parts of the city closer to water bodies in the case of floods, closer to forests in the case of wildfires and closer to the eye in the case of tornadoes. However, the destruction of landslides and wildfires measured in economical damage is little compared to earthquakes and hurricanes, see Figure 6.3.

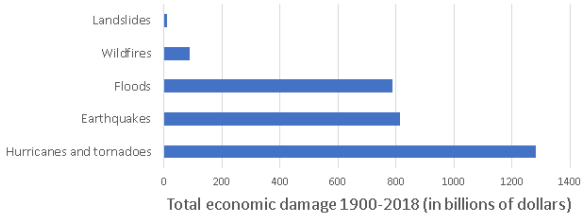


Figure 6.3: Total worldwide economic damage per disaster type (1900-2019) [28].

The same goes for tornadoes. Economically, tornadoes cause about a tenth as much damage per year as hurricanes [7]. Hence, floods are the only type of natural disaster that cause i) a lot of economical damage and debris and ii) an unequal spread of debris across the city. For simplification reasons, we decided not to take this into account. We therefore consider the intensities of disasters to be equally spread over the city. We do however consider the first cause for an unequal debris spread: difference in population density across a city. We will distinguish two types of instances: cities where every street has more or less the same debris amount and cities where streets have debris amounts corresponding to their population. We argue that more populated streets are more prone to produce more debris. An estimation of the population of a street is made using two data points for every street we obtained from Open Street Map: its length and its type. Clearly, the length of a street is an indicator of the population of the street. Street types are provided by Open Street Map too and indicate the importance of a road in the street network ranging from highways to residential roads. They can give an indication of the population of a street under the assumption that, in cities, less important and residential roads have a higher population than important roads like highways. The type of a street results in a debris multiplier for that street. All street types with their description and multipliers are shown in Table 6.1.

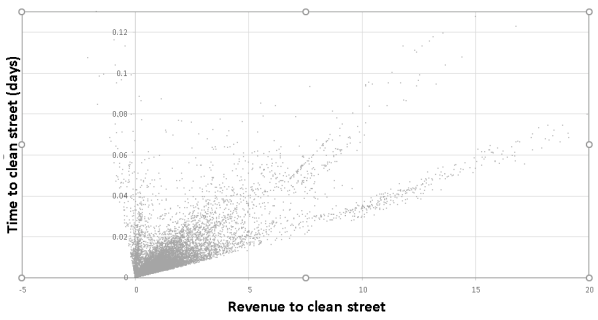
Number	Name	Description	Multiplier τ_e
1	Motorway	Highways	0.1
2	Primary	Most important road that aren't highways	0.2
3	Secondary	The next most important roads	0.3
4	Tertiary	The next most important roads	0.4
5	Residential	Roads with housing	1
6	Other	Miscellaneous roads like dirt tracks	0.1

Table 6.1: Roadtypes with names, description and multipliers.

The second question regarding the edge weights is hence tackled by creating two types of instances. One where edge weights are more or less the same across streets and one where edge weights depend on the length and type of the street.

The third question regarding the weights we are trying to answer in our experimental design deals with the correlation between the time and profit corresponding to each street. One might say that streets with a lot of debris take more time to clean and are more profitable. Therefore a positive correlation between the two seems logical. In reality however, it is more complicated due to the different types of debris. Some streets might be full of non-recyclable debris and cause large cleaning times and negative profits. To show the relation between time and profit we analyzed data from a simulated disaster in Puerto Rico [31]. In Figure 6.4 the times and profits of all streets in Puerto Rico are shown for four different contractors.

(a) Contractor 1.



(b) Contractor 2.

(c) Contractor 3.

(d) Contractor 4.

Figure 6.4: Correlation between expected time and profit to clean a street for four different contractors. Each dot represents a street in Puerto Rico after a simulated hurricane.

In total, 10 contractors were analyzed in the Puerto Rico case. For each contractor the correlation between the time and profit values is shown in Figure 6.5.

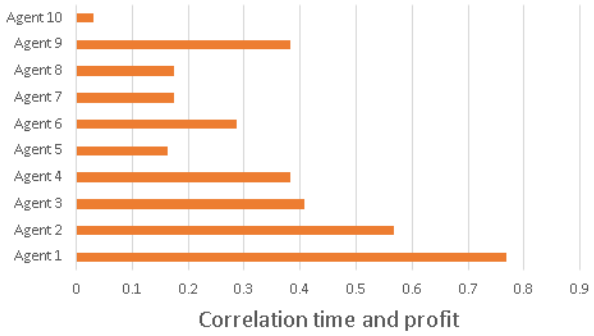


Figure 6.5: Correlation coefficients of time and profit for all streets of Puerto Rico after a simulated disaster for 10 contractors.

Figures 6.4 and 6.5 indicate that time and profit are correlated in some cases and uncorrelated in others. To investigate the impact of this variability on the performance of our algorithm we distinguish two cases: one where time and profit are correlated and one where they are not.

We have discussed the three questions that we will answer regarding the edge weights by making three choices. One choice for every question: i) symmetric or asymmetric contractors, ii) equal or unequal spread of debris and iii) correlated or uncorrelated time and profit values. This gives us $2 \times 2 \times 2 = 8$ weight settings S_i for $i = 1, 2, \ldots, 8$. All weight settings have a specific combination of the three choices. In Table 6.2 it is shown how we obtain the weights for each of the eight weight settings.

Contractors	**Debris spread**	**Time and Profit**	**Setting**	$w_{e,k}^{(1)}$	$w_{e,k}^{(2)}$
Symmetric	Equal	Uncorrelated	S_1	$\sim \mathcal{N}(5,1)$	$\sim \mathcal{N}(5,1)$
		Correlated	S_2	$\sim \mathcal{N}(5,1)$	$\sim \mathcal{N}(w_{e,k}^{(1)},1)$
	Unequal	Uncorrelated	S_3	$\sim \mathcal{N}(l_e\tau_e, 0.2 l_e\tau_e)$	$\sim \mathcal{N}(5,1)$
		Correlated	S_4	$\sim \mathcal{N}(l_e\tau_e, 0.2 l_e\tau_e)$	$\sim \mathcal{N}(w_{e,k}^{(1)}, 0.2 l_e\tau_e)$
Asymmetric	Equal	Uncorrelated	S_5	$= \frac{\gamma_{e,k}^{(1)}}{c_k}, \gamma_{e,k}^{(1)} \sim \mathcal{N}(5,1)$	$\frac{\gamma_{e,k}^{(2)}}{c_k}, \gamma_{e,k}^{(2)} \sim \mathcal{N}(5,1)$
		Correlated	S_6	$= \frac{\gamma_{e,k}^{(1)}}{c_k}, \gamma_{e,k}^{(1)} \sim \mathcal{N}(5,1)$	$\frac{\gamma_{e,k}^{(2)}}{c_k}, \gamma_{e,k}^{(2)} \sim \mathcal{N}(\gamma_{e,k}^{(1)},1)$
	Unequal	Uncorrelated	S_7	$= \frac{\gamma_{e,k}^{(1)}}{c_k}, \gamma_{e,k}^{(1)} \sim \mathcal{N}(l_e\tau_e, 0.2 l_e\tau_e)$	$\frac{\gamma_{e,k}^{(2)}}{c_k}, \gamma_{e,k}^{(2)} \sim \mathcal{N}(5,1)$
		Correlated	S_8	$= \frac{\gamma_{e,k}^{(1)}}{c_k}, \gamma_{e,k}^{(1)} \sim \mathcal{N}(l_e\tau_e, 0.2 l_e\tau_e)$	$\frac{\gamma_{e,k}^{(2)}}{c_k}, \gamma_{e,k}^{(2)} \sim \mathcal{N}(\gamma_{e,k}^{(1)}, 0.2 l_e\tau_e)$

Table 6.2: An overview of the eight weight settings with their corresponding values of $w_{e,k}^{(1)}$ and $w_{e,k}^{(2)}$. l_e denotes the length of street e, τ_e denotes the road type of street e and c_k denotes the efficiency factor of a contractor k.

Drawing the weight values from their distribution as shown in Table 6.2 results in the scatter plots of time and profit values shown in Figure 6.6.

Agents	Debris spread	Time and profit	Setting
Symmetric	Equal	Uncorrelated	S_1
Symmetric	Equal	Correlated	S_2
Symmetric	Non-equal	Uncorrelated	S_3
Symmetric	Non-equal	Correlated	S_4
Asymmetric	Equal	Uncorrelated	S_5
Asymmetric	Equal	Correlated	S_6
Asymmetric	Non-equal	Uncorrelated	S_7
Asymmetric	Non-equal	Correlated	S_8

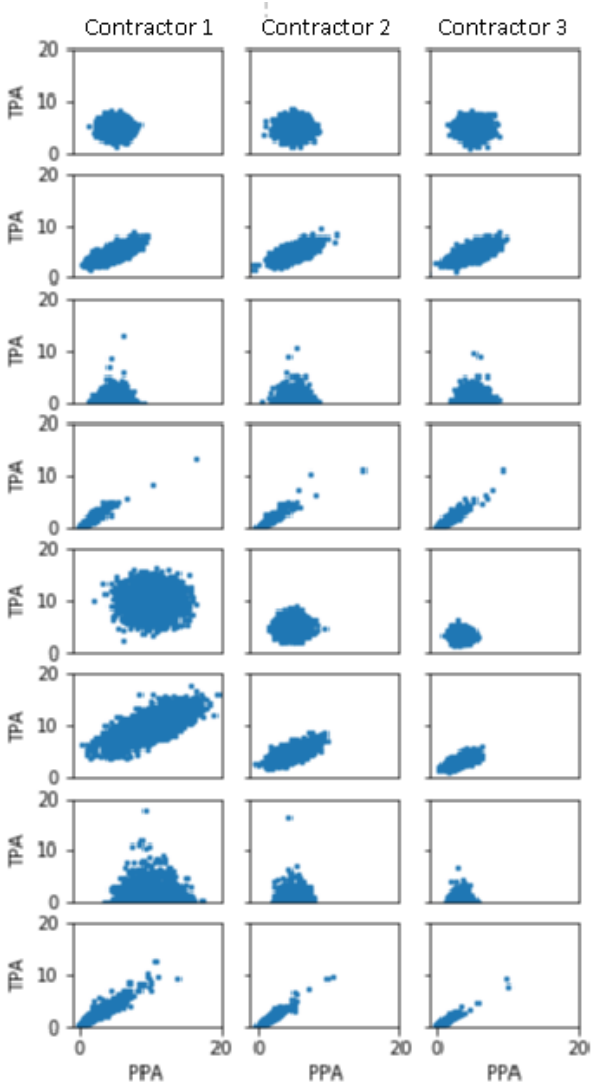
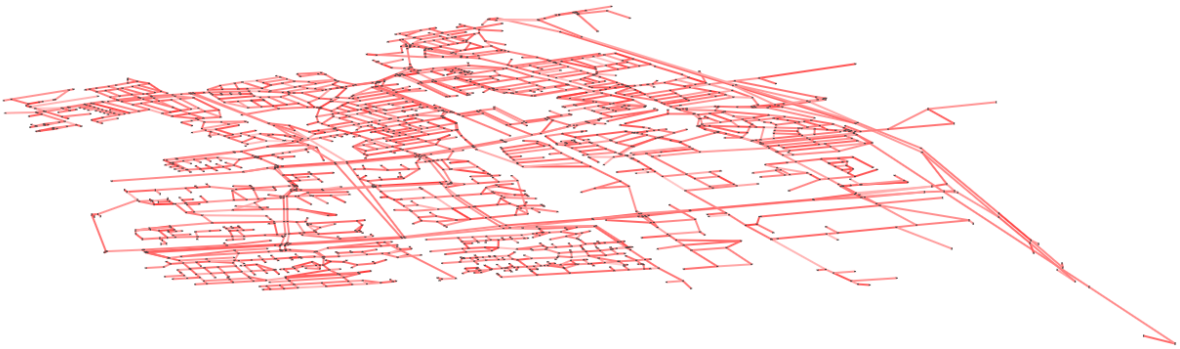
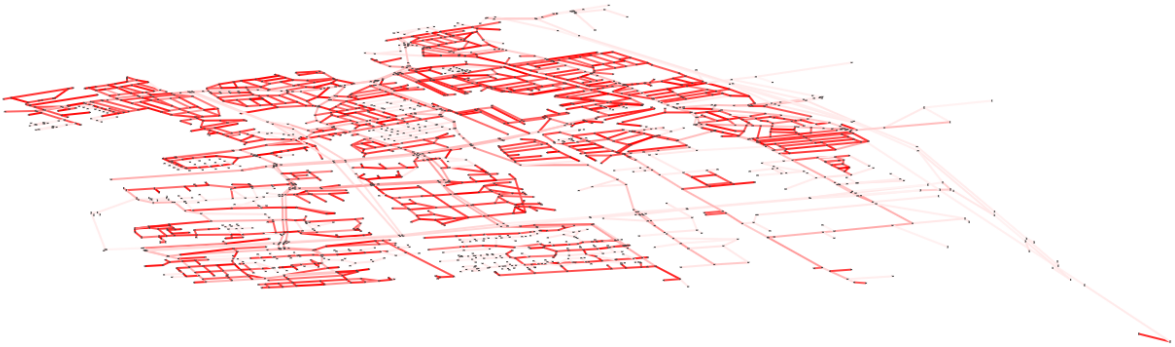


Figure 6.6: Scatter plots of time and profit weights for 3 contractors for each weight setting. Every dot represents a street in the city of Delft, the Netherlands. TPA represents the value of $w_{e,k}^{(1)}$ and PPA represents the value of $w_{e,k}^{(2)}$.

Figure 6.6 shows us that we chose the parameters in Table 6.2 correctly. First, observe the difference between the symmetric and asymmetric plots. In the case of symmetric contractors the plots look similar for each contractor. In the case of asymmetric contractors however, we see that contractor 3 has lower time and profit values than contractor 1. Secondly, we observe the difference between uncorrelated and correlated settings. In the uncorrelated cases we observe no relation between time and profit values. The correlated case however, clearly shows a positive relation: streets with a higher cleaning time, have a higher profit. The difference between an equal and a non-equal spread of debris does not clearly show from these graphs. Figure 6.7 does show this. Here, the amount of debris in the city of Delft, the Netherlands for setting 1 and setting 3 is shown.



(a) Debris spread in setting 1.



(b) Debris spread in setting 3.

Figure 6.7: Debris spread in kg per meter on the driving network of the city of Delft, the Netherlands for weight setting 1 and weight setting 3.

In Figure 6.7a one can observe that the debris is equally spread along the city. Each street has roughly the same amount of debris. In Figure 6.7b we see that the debris is clustered in certain areas. These areas are the residential areas. Bigger roads like highways have less debris.

In the above we have discussed three types of choices we make for each instance: the number of contractors, the type of graph and the weight setting. By making a choice in each category we obtain what we call an instance. An example of an instance can be "5 contractors, Miami, Florida, weight setting 3". Since we have 2 sizes of our contractor set, 9 different cities and 8 different weight settings, we obtain $2 \times 9 \times 8 = 144$ instances. Now it is time to run our algorithm on each of these instances. How we do this is described in the following section.

6.2. Running algorithms

In the previous section we explained how we create our instances. Now, we show how we run the final UM algorithm, described in Algorithm 9 in Section 5.4, on each of these instances. This is done the easiest by taking one instance as an example. Let us suppose our instance has 5 contractors, has Nijmegen, the Netherlands as its graph, and weight setting 4.

We first calculate the optimal value of the time, T_{opt}, when the other objectives are ignored. We ask and answer the following question: How fast can we clean the city when profit and niceness are ignored? In other words, we solve $P_1(G)$ without constraints (3.1b), (3.1c), (3.1e) and (3.1f). We do this with Gurobi [33], an MIP solver, and run until we obtain an gap between the lower bound and objective value of less than 0.1%. This is done in seconds. In our specific instance of 5 contractors, Nijmegen, the Netherlands and weight setting 1 the optimal time is 564 hours.

We then calculate the optimal value of the profit,P_{opt}, when the other objectives are ignored. We ask and answer the following question: what is the profit of the least earning contractor, when the time and niceness are ignored? In other words, we solve $P_1(G)$ without constraints (3.1a), (3.1c), (3.1d) and (3.1f). Again, we run until we obtain an gap between the upper bound and objective value of less than 0.1%. In our case we find an optimal profit of 974 dollars per asset.

We now compute a Pareto Front of the time and profit values while ignoring the niceness. We ask and answer the following question: what are the best time and profit values when we ignore the niceness? We pick ten values for the time ranging from 50% to 100% of its optimum. We then optimize the profit value while bounding the time value. Once for every 10 time values in this range. We run until we obtain a gap between the lower bound and objective value of less than 1%. In other words, we solve $P_1(G)$ without constraints (3.1a), (3.1c) and (3.1f), while bounding (3.1d). In Figure 6.8 the obtained Pareto Front is shown for our case.

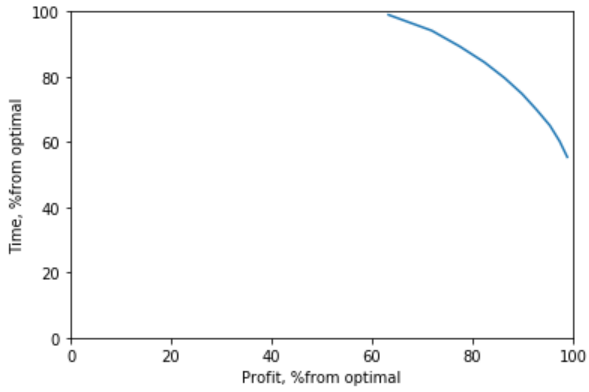


Figure 6.8: Pareto Front of time and profit for weight settings 4 for the city of Nijmegen (5500 streets) in The Netherlands with 5 contractors.

Figure 6.8 shows that, as expected, a better time, leads to a worse profit and vice-versa. For our instance, we now have several solutions x on the time-profit Pareto Front. All of these solutions x are still chaotic. We now choose one x on the Pareto Front which we are going to make "nice". The solution x we pick is the solution with the highest value of $\min\{time, profit\}$, where time and profit for each solution are expressed in percentage of the optimal. This solution corresponds to the point on the line in Figure 6.8 closest to the upper right corner. In our instance the solution we pick has a profit value of 799 dollar per asset, 82% of the optimal profit. It has a time value of 655 hours, 84% of the optimal time. Hence, our solution has decent time and profit objective values. However, it is still chaotic. This is where the final UM algorithm, described in Algorithm 9 comes in.

We run this algorithm for three values of α: 0.8, 0.7 and 0.6. α resembles the allowed relaxation of the time and profit values, while we reduce the chaos. The upper bound T_{max} for the time is chosen such that $\frac{|T_{max}-T_{opt}|}{T_{opt}} = \alpha$. Similarly, the lower bound P_{min} for the profit is chosen such that $\frac{|P_{min}-P_{opt}|}{P_{opt}} = \alpha$. With these bounds in mind, we solve $P_1(G)$ in two ways: with the MIP solver Gurobi and with our final UM algorithm.

Let us start with explaining how we solve $P_1(G)$ with the MIP solver Gurobi. The MIP we give to Gurobi is $P_1(G)$ without the objectives (3.1a) and (3.1b) and bounding constraint (3.1d) with the time bound and constraint (3.1e) with the profit bound. We run until we reach an MIP gap of less than 1% or until we reached a time limit of 5 hours. We save every solution that Gurobi outputs on the way.

We solve the same instance with our final UM algorithm. To do this we run Algorithm 9. This algorithm repeatedly solves the program P_{UM}, described in Expression (5.1), approximately. In P_{UM}, we use for B_1 the upper bound for the time T_{max}. For B_2 we use the the lower bound for the profit P_{min}. We run every iteration of the starting and ending UM algorithm until an MIP gap of 1% or the time limit of 200 seconds is reached.

The entire code for the final UM algorithm and the MIP-solver can be found on Github by searching for the user "lucasvogels33". We use Gurobi 8.1.0 which we run in Python 3.7. All code is run on a HP Notebook with

model number 15-ay164nd with a Intel Core i5-7200U processor.

For our instance of 5 contractors, on Nijmegen, the Netherlands with weight setting 4, we will now have six solutions of $P_1(G)$, since we have for each of the three values of α one solution provided by the MIP solver and one solution provided by our algorithm.

We have introduced our problem $P_1(G)$. We have come up with structural results for it and designed an algorithm based on of these structural results. We explained how we are testing this algorithm. We are now ready to see the results of our experiments. Does the algorithm perform?

7 Results

In this chapter we will discuss the results of the experiments described in Chapter 6. We will first compare the results of the final UM algorithm with the results of the MIP solver. We will then see how the performance of the UM algorithm depends on the type of instance it is ran on.

7.1. UM algorithm vs MIP solver

In this section we will show that the final UM algorithm as described in Section 5.4 completely outperforms the MIP solver when solving $P_1(G)$.

In Chapter 6 we said that we tried solving $P_1(G)$ to a 1% optimality gap using Gurobi, an MIP solver. We instructed the solver to also output all solutions that it finds along the way. Even if those solutions have a worse optimality gap. There was not one single instance for which the MIP solver gave a solution with an optimality gap less than 1%. Neither were there any mid-way solutions with a worse optimality gap found. The cities are simply to large for an MIP solver to handle. Even Miami, Florida, the smallest city tested with 12.500 streets, turned out to be too big to be solved to any accuracy in 5 hours. Now imagine Philadelphia with 40.000 streets. A simply impossible task for a regular MIP solver. To test whether the MIP solver was correctly programmed, we ran the MIP solver on small cities. In Figure 7.1 the solution of the MIP solver on the small city of Duncan in Canada is shown.

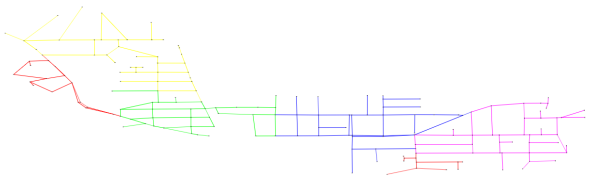
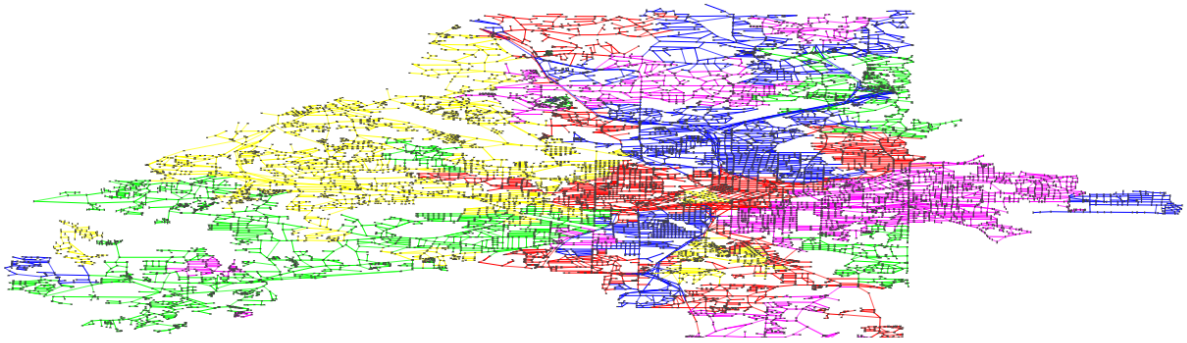


Figure 7.1: The city of Duncan, Canada, solved by the MIP solver for 5 contractors, a weight setting 1, an $\alpha = 0.4$

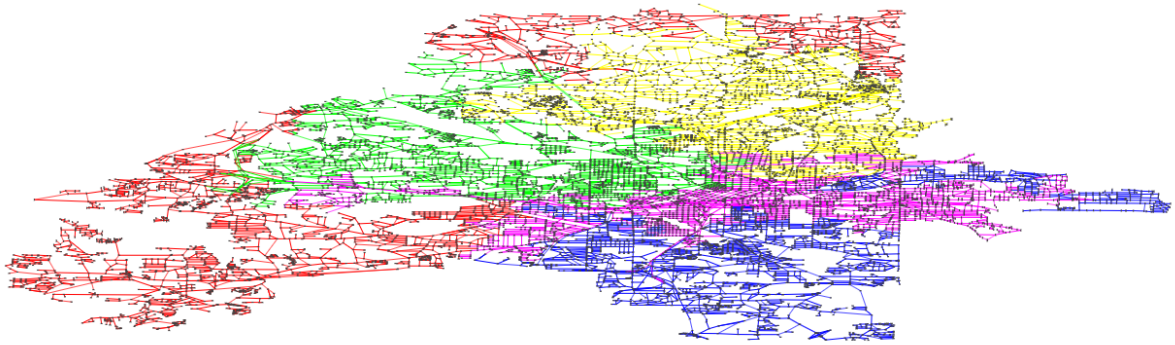
Running the MIP solver on Duncan and other small cities shows us that it does in fact work. It can successfully produce good solutions to $P_1(G)$ for small instances. However, as the cities get bigger and the number of edges increases, the complexity of the problem explodes and the MIP solver becomes useless. Any city big enough to have a debris cleaning process in place has too many streets for an MIP solver. There is need for another way to solve the debris management problem.

This is where our algorithm (Algorithm 9) comes in. It produced good solutions for all cities, weight settings and number of contractors. And, astonishingly, it did so in under one hour for all instances. Most instances were even solved under 15 minutes. Before we dive into the analysis of the results, let us show the power of the final UM algorithm, by example. In Figure 7.2 we see solutions for different values of α produced by the final UM algorithm for Atlanta, Georgia with 5 contractors under weight setting 1.

(a) $\alpha = 0.8$. Hence time and profit are 80% from their optimal value.



(b) $\alpha = 0.7$. Hence time and profit are 70% from their optimal value.



(c) $\alpha = 0.6$. Hence time and profit are 60% from their optimal value.

Figure 7.2: The city of Atlanta, Georgia, solved by the final UM algorithm for 5 contractors and weight setting 1. Solutions are shown for three different values of α.

The solutions in Figure 7.2 were obtained within just 10 minutes. A time unthinkable for an MIP solver. Note how the solutions look less chaotic as α decreases. This makes sense, since a lower α corresponds to looser bounds for the time and profit objectives.

7.2. Performance per category

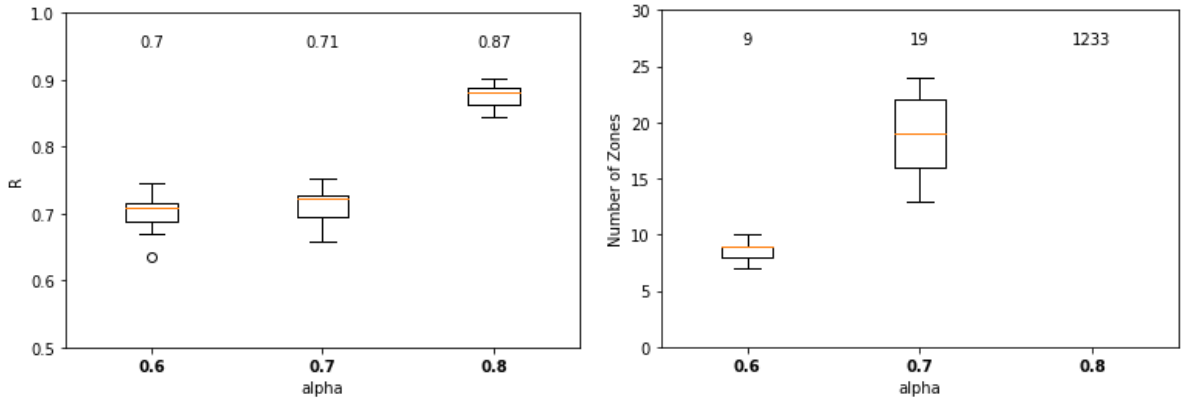
We observed that the final UM algorithm (Algorithm 9) outperforms an MIP solver. From now on we will only focus on the performance of the final UM algorithm. It will be shown how its performance depends on the amount of contractors, the city and the weight setting.

In Chapter 6 we explain that for every city, weight setting and number of contractors, we obtain three solutions. One for every value of α. We can measure the chaos of these solutions in two ways: the number of zones and the sum of span. However, when we compare different cities, this second measure becomes troublesome. Big cities automatically have a bigger sum of span, because they have more nodes. We define a new

measure R to correct for this.

$$R := \frac{\text{sum of span}}{\text{edges}}$$

We can now measure the chaos of every solution with R and the number of zones. We will start with one specific case to explain how we will present the results. This is the case of weight setting 1 and a contractor amount of 5. In Figure 7.3 the results for this case are shown.



(a) Choas metric R

(b) Chaos metric number of zones

Figure 7.3: The results of the experiments for the case of weight setting 1 and 5 contractors. Figure 7.3a shows the chaos metric R and Figure 7.3b shows the number of zones. Both figures contain 3 box plots corresponding to $\alpha = 0.6, 0.7$ and 0.8. Every boxplot shows the spread of datapoints. Each datapoint represents the value of the chaos measure for a certain city and α under weight setting 1 and contractor amount 5. The number above every boxplot represent the mean of the data points in that boxplot.

Let us explain how one should read Figure 7.3a. It shows that, when we relax time and profit to be 60% ($\alpha = 0.6$) of their optimal values, the chaos measure R was between 0.65 and 0.75 for all nine cities with an average of 0.7. When we tighten the time and profit bounds to 70% ($\alpha = 0.7$), the chaos measure R is slightly higher. The chaos metric R of the nine cities varies again between 0.65 and 0.75. The average is 0.71. When we further tighten the time and profit bounds to 80% ($\alpha = 0.8$) of their optimal value, we see that the average chaos measure R increased to 0.87.

Figure 7.3b can be interpreted similarly. When we allow time and profit to be 60% ($\alpha = 0.6$) of their optimal values, the number of zones was low for all cities. It varied between 7 and 10 with an average of 9. When we tighten the time and profit bounds to 70% ($\alpha = 0.7$), the number of zones in the solution is higher for all cities. It varies between 12 and 23 with an average of 19 zones. When we further tighten the time and profit bounds to 80% ($\alpha = 0.8$) of their optimal value, we see that the number of zones explodes to an average of 1233 zones. The boxplot is not shown because it is outside the range of the y-axis. We can conclude that an α of 0.8 results in a completely chaotic assignment in this case.

Figure 7.3 shows the results for weight setting 1 and 5 contractors. We can produce the same boxplots for all eight weight settings and for 5 and 10 contractors. The results are shown in Figure 7.4.

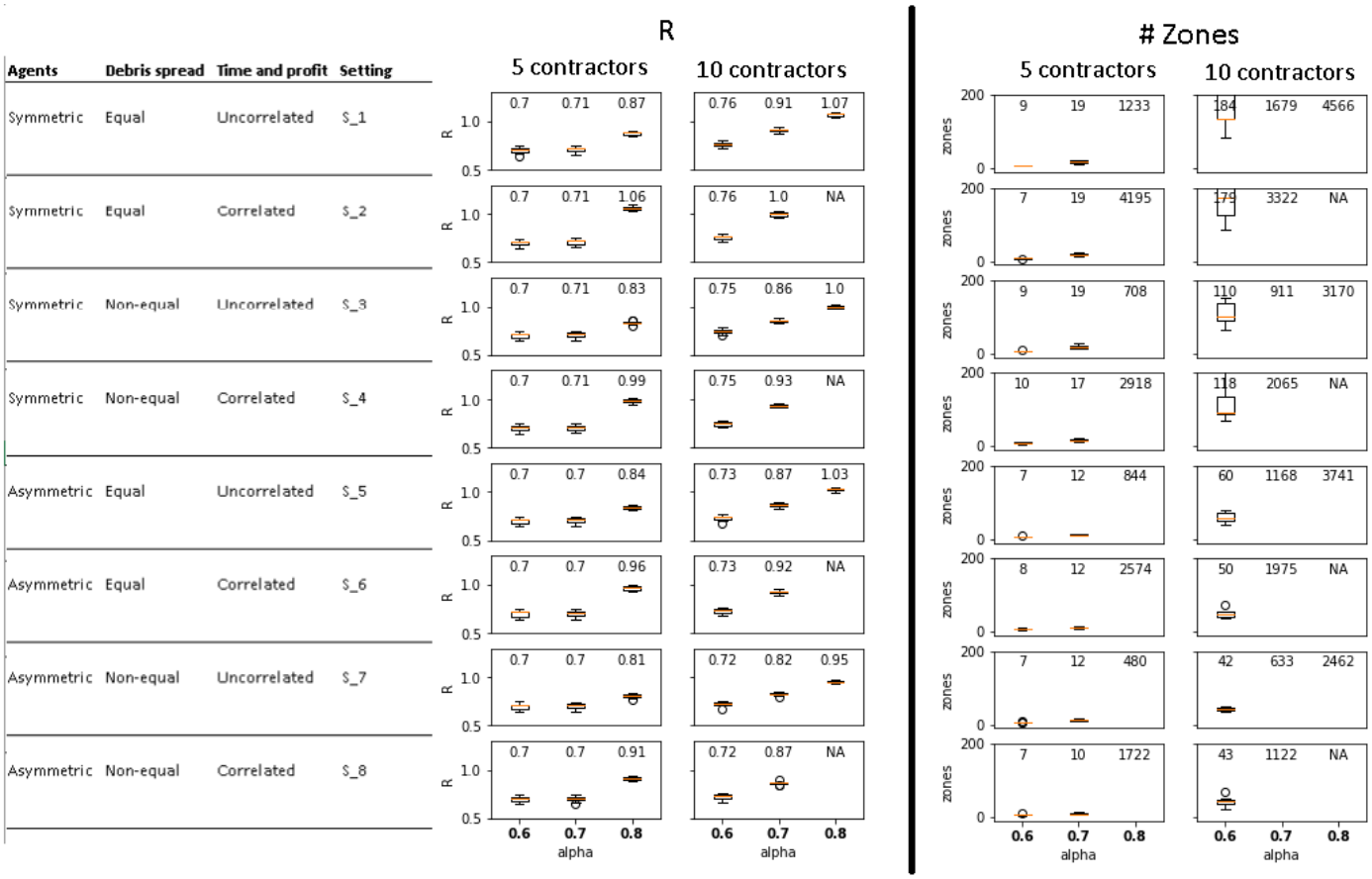


Figure 7.4: The results of the experiments. We measured niceness in two ways: using $R := \frac{\text{sum of span}}{\text{edges}}$ and using the number of zones. Every combination of weight setting S_i and contractor amount has its own figure. Every figure contains 3 box plots corresponding to $\alpha = 0.6, 0.7$ and 0.8. Every box plot shows the spread of datapoints. Each datapoint represents the value of the chaos measure for a certain city, α, weight setting and contractor amount. The number above every box plot represents the mean of the data points in that box plot.

Note that for 10 contractors combined with weight setting 2,4,6 or 8 and $\alpha = 0.8$ there were no feasible solutions found by the algorithm. This is shown in Figure 7.4 by "NA" for non-applicable. In all other cases solutions were found for every city. We will see why this is the case later.

Figure 7.4 tells us that the algorithm performs performs well. When we have 5 contractors, the algorithm creates non-chaotic partitions at 70% of the optimal time and profit values. This holds across all weight settings and all nine cities.
We will now interpret Figure 7.4 for each of the three parameter classes discussed in Chapter 6: the number of contractors, the weights $w_{e,k}^{(i)}$ and the graph structure G. For each parameter class we will i) repeat the questions asked in Chapter 6, ii) answer those questions using the observations from our experiments and iii) give recommendations to local governments of cities prone to natural disasters.

7.2.1. The amount of contractors

Questions

In Chapter 6 we asked the following question: what is the impact of the number of contractors on the performance of the final UM algorithm? Let us look at the results and answer the question.

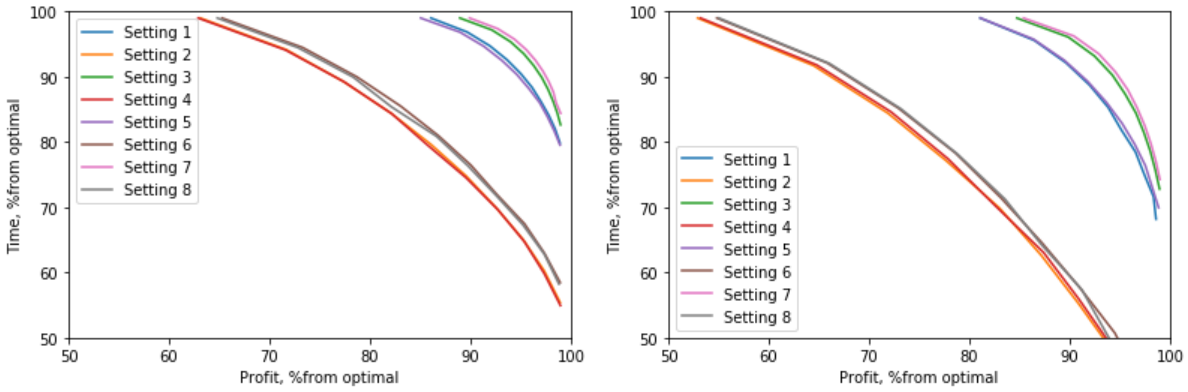
Observations

The algorithm works better when less contractors are involved.

Figure 7.4 tells us that, when time and profit are 60% from their optimal values ($\alpha = 0.6$), all instances with 5 contractors are reduced to 10 or less zones. Tightening the time and profit bounds to 70% still gives non-

chaotic solutions with on average less than 20 zones for all weight settings. The algorithm has more difficulties with 10 contractors. With 10 contractors, an α of 0.6 still results, on average, between 43 and 184 zones. Tightening the time and profit bounds to 70% ($\alpha = 0.7$) or 80% ($\alpha = 0.8$) results in completely chaotic solutions with more than 600 and sometimes even thousands of zones. For weight settings 2, 4, 6 and 8 the algorithm does not even output a solution.

Why does the algorithm perform better when there are less contractors involved? The answer can be found in Figure 7.5 that shows the Pareto fronts of the time and profit objectives for each weight setting for the city of Nijmegen, the Netherlands. It shows the optimal time and profit values when we completely ignore the niceness objective. Figure 7.5a shows the Pareto fronts in the case of 5 contractors and Figure 7.5b shows them in the case of 10 contractors.



(a) 5 contractors

(b) 10 contractors

Figure 7.5: Pareto fronts for the time and profit objectives for the city of Nijmegen (5500 streets)

We observed similar Pareto Fronts for all other cities, regardless of their size. Notice that in the case of 10 contractors the Pareto Fronts are closer to the left bottom of the graph than in the case of 5 contractors. This means that, when we ignore the niceness objective, we achieve less optimal time and profit values when we have more contractors. In the final UM algorithm we relax the time and profit objectives to 80%, 70% and 60% of their optimal. When we have only 5 contractors this relaxation creates more room to improve the niceness than in the case of 10 contractors. Figure 7.5b shows that for 10 contractors a relaxation of 80% is not even feasible for weight settings 2, 4, 6 and 8. This corresponds to the observation that for these instances the final UM algorithm does not output feasible solutions.

Recommendations

We saw that our algorithm performs worse when hiring more contractors. Does this mean that policy makers should avoid hiring many contractors to do clean-up operations? No, not necessarily. We will show that this depends on the preferences of the policy makers by showing the effect of more contractors on time, profit and niceness. We limit ourselves to weight setting 1. Other weight settings show similar behaviour.

Clearly, hiring more contractors results in a lower total cleaning time, since there are more resources to clean the same amount of debris. But how much lower? Figure 7.6 answers this question.

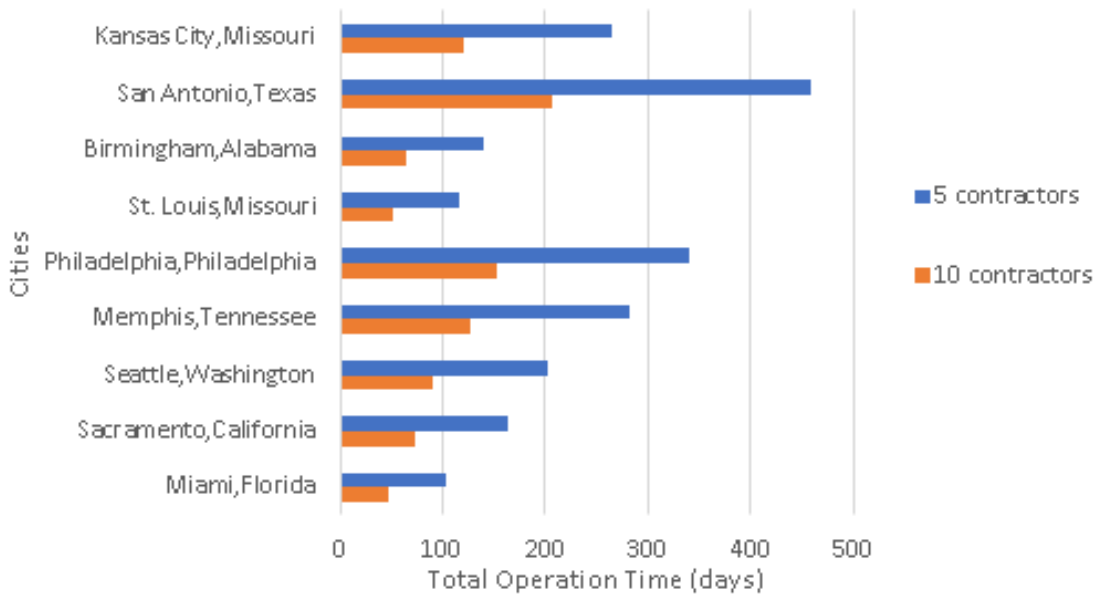


Figure 7.6: Total operation times for nine cities, for 5 and 10 contractors and weight setting 1.

Figure 7.6 shows that the total cleaning time approximately halves when increasing the contractor amount from 5 to 10. Strikingly, the cleaning time decreases by the same percentage for all nine cities: 55%. Hence doubling the amount of contractors, reduces the operation time by more than halve.

What about the profit? It is obvious too that hiring more contractors results in a lower profit for the poorest contractor, since there are more contractors to share the same amount of money. But how much lower? Figure 7.7 answers this question.

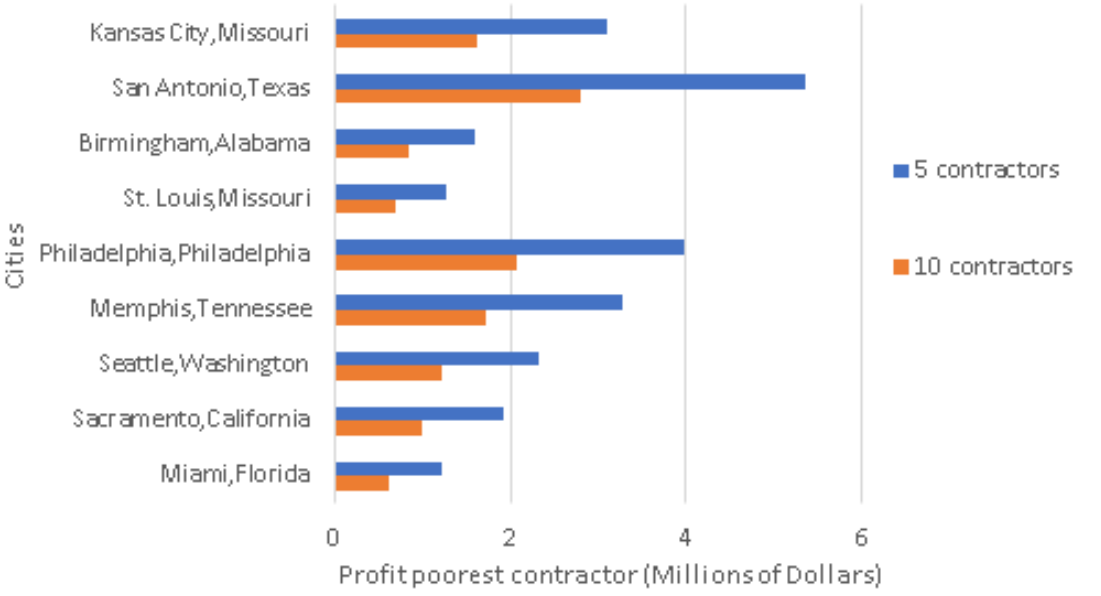


Figure 7.7: Profit of the poorest contractor, for 5 and 10 contractors and weight setting 1.

Figure 7.7 shows that the profit for the poorest contractor approximately halves when increasing the contractor amount from 5 to 10. In fact, it decreases between 49% and 45% for all nine cities. Hence doubling the

amount of contractors, reduces the profit of the poorest contractor by less than halve.

Figures 7.6 and 7.7 indicate that it is beneficial for the time and profit objectives to increase the amount of contractors. This might seem counter intuitive with Figure 7.5, where we saw that an increase in contractors causes less optimal time and profit values. However, in Figure 7.5 time and profit are measured as a percentage from their optimal value, whereas in Figures 7.6 and 7.7 the absolute values of time and profit are given. Hence, increasing the amount of contractors leads to worse time and profit values, when they are measured as a percentage from their optimal, but to better absolute time and profit values.

We conclude that doubling the contractor fleet results in a reduction of the time of more than 50%, a reduction of the profit of less than 50% and a more chaotic solution. It depends on the objective that the policy maker prioritizes, whether it is advisable to increase the amount of contractors.

7.2.2. The weight setting

Questions

In Chapter 6 we asked the following questions regarding the edge weights:

1. What is the impact of symmetric versus asymmetric contractors on the performance of our algorithm?
2. What is the impact of an equal spread of debris versus a non-equal spread of debris on the performance of our algorithm?
3. What is the impact of a correlated time and profit and a non-correlated time and profit?

Let us look at the results and answer these questions.

Observations

The final UM algorithm (Algorithm 9) works better for asymmetric contractors than for symmetric contractors. It works equally well for an equal and a non-equal spread of debris. Lastly, it works better for uncorrelated time and profit weights than for correlated time and profit weights.

Figure 7.4 tells us that when we have asymmetric contractors, we obtain better solutions. In the case of 10 contractors and an α of 0.6, we see that asymmetric contractors partition the graph in 30-60 zones, while symmetric contractors end up with 100-156 zones. Moreover, in the case of 5 contractors and an α of 0.7, we see that asymmetric contractors partition the graph in 10-12 zones, while symmetric contractors end up with 17-19 zones. This might be explained by the fact that a team of diverse contractors is better equipped to deal with diverse debris. This hypothesis is supported by the fact that, with 10 contractors and $\alpha = 0.6$, asymmetric contractors perform better when there is a non-equal spread of debris (around 40 zones) than when there is an equal spread (around 50-60 zones).

There is no real difference in the performance of the algorithm between equal and non-equal debris spread. The chaos metrics R and the number of zones are very similar for both 5 and 10 contractors and all values of α.

Lastly, we observe a difference in performance between uncorrelated and correlated time and profit weights. The final UM algorithm performs better when the time and profit weights are uncorrelated. This effect only occurs for higher values of α. Figure 7.4 shows that for $\alpha = 0.6$ there is no difference between uncorrelated and correlated time and profit weights. When we increase α to 0.7 we see a difference occurring in the case of 10 contractors. For $\alpha = 0.8$ the difference is biggest. For 5 contractors, the weight settings that are correlated show higher chaos metrics than uncorrelated weight settings. For 10 contractors, correlated weight settings even become infeasible. This effect can be explained by the Pareto fronts of the time and profit objective shown in Figure 7.5. The correlated weight settings 2,4,6 and 8 have their Pareto fronts further away from their optimal values than the uncorrelated weight settings 1,3,5 and 7. Therefore, instances with these weight settings have less “room” to improve their chaos metric, especially for high values of α.

Recommendations

The observations show that the characteristics of the edge weights matter for the duration, profit and chaos of the cleaning schedule.

Governments often know before a disaster whether the time and profit related to cleaning every street will be correlated or uncorrelated and whether the spread of the debris will be equal or non-equal. When they are correlated they should prepare for more chaotic debris collection schedules. Moreover, when a non-equal spread of debris is expected, policy makers need to hire a diverse fleet of contractors.

7.2.3. The city

Questions

In Chapter 6 we asked the following question regarding the structure of the street plan of a city: does the city street plan have an impact on the performance of the algorithms? Let us look at the results and answer these questions

Observations

We see a striking lack of variety in the results across the cities.

We tested our algorithm on nine American cities. Their sizes vary from 12000 streets to 40000 streets. Some cities have rivers, some do not. Some are cities are coastal, some are not. Moreover, the street plans of the cities vary greatly. It is therefore striking that the algorithm performs very similarly across these cities. Figure 7.3 shows that for weight setting 1, 5 contractors and $\alpha = 0.6$ all cities have their chaos metric R between 0.65 and 0.75. The number of zones varies between 7 and 10. The same goes for different values of α. Figure 7.4 shows that for all weight settings, contractor amounts and values of α, the nine cities behave very similar. Moreover, in Subsection 7.2.1, we observed that a doubling of the amount of contractors from 5 to 10 resulted in the same decrease in operation time for all cities: 55%

The robustness of the final UM algorithm indicates that it will produce good results on all cities worldwide.

8

Conclusion

This thesis discussed the problem of Unrelated Unconnected Multi Constrained Graph Partitioning (UUMCGP). Here, an edge set E of a graph G is partitioned into subsets with three objectives in mind: i) balance the total amount of the first weight among the subsets, ii) balance the total amount of the second weight among the subsets and iii) create a non-chaotic partition. This problem has applications in the management of debris after natural disasters and in assigning tasks to computers in a distributed network. In the introduction we asked four questions that we will answer now.

Can we define the UUMCGP as an MIP?
Yes. We are the first to formally define and describe the mixed-integer program of UUMCGP.

Can we solve this MIP in polynomial time?
No. We show that solving this multi-objective problem is hard in practice. Even when we focus only on the first objective or only on the second objective, the problem is NP-hard. However, we discuss four cases for which we can produce solutions that get arbitrarily close to the optimal solutions for big enough graphs.

Can we come up with good approximate solutions to this MIP using an algorithm?
Yes, we are the first to do so. We first designed an algorithm for a different mixed-integer program: the Multi-Resource Generalized Assignment Problem (MRGAP). This algorithm solves the MRGAP, while violating the constraints only by a certain amount. This novel result expands the field of multi-resource generalized assignment problems. We use this result to develop an algorithm that produces good approximate solutions to the general case of UUMCGP.

How does this algorithm perform in real-life cases of debris management?
It does well in these cases, producing fast and profitable cleaning schedules for nine disaster prone American cities. When there are five contractors, the algorithm outputs non-chaotic partitions at 70% of the optimal time and profit values. The results are very similar across the nine cities, what suggests that the algorithm could be applied on any city worldwide. It is the first algorithm that can produce cleaning schedules of any city without the help of a human.

9 Further Research

In this chapter we will discuss the shortcomings of this thesis and suggest topics for future research.

9.1. Applicability of UUMCGP to the debris management problem

Shortcomings

In Chapters 1 and 2 we abstract the debris management problem to Unrelated Unconnected Multi-Constrained Graph Partitioning (UUMCGP). In Chapter 3 we formulate the mixed-integer program (MIP) $P_1(G)$ for UUMCGP. While doing so, we leave out two major aspects of the debris management problem that hurt the applicability of UUMCGP to debris collection: the role of facilities and the lack of information on the time and profit values.

Facilities play an important role in the collection of debris. However, for simplification reasons, they are left out in the formulation of UUMCGP. This hurts the applicability of UUMCGP to the debris management problem.

UUMCGP assumes that the time and profit corresponding to cleaning every street is known for every contractor. In other words, it assumes that the values of $w_{e,k}^{(i)}$ in $P_1(G)$ are deterministic. One could argue whether this is realistic. In the aftermath of a disaster there are many uncertainties. An estimation of the time a contractor would take to clean a pile of debris is uncertain. UUMCGP does not allow for this uncertainty. Again, this hurts the applicability of UUMCGP to the debris management problem.

Possible solutions

There already exists research in debris management involving facilities. Stilp considers UUMCGP with facilities [31]. However, the author focuses on the development of a human operated computer tool to design cleaning schedules. It is valuable to know whether the final UM algorithm can be updated to approximately solve UUMCGP with facilities. This could result in a fully automated algorithm to solve the debris management problem with facilities.

There is limited research available that considers the collection phase of debris management with probabilistic debris weights. Improvement in this area would be of great value in designing better disaster response tools.

9.2. Vertex connectivity

Shortcomings

In Subsection 4.1.3 four theorems are given. They show that when we increase the size of the edge set E, we can obtain solutions that are arbitrarily close to the optimal solutions. We obtain these results because we restrict the weights, the number of agents and the structure of the graph G. The restrictions on the number of agents are realistic. Apart from Theorem 2, where we set the number of agents to 2, we do not restrict the agents. The values for the edge weights are chosen realistically too. All four theorems resemble edge weights

that could occur in real-life too. However, the structure of the graphs treated in these theorems is limited to r-vertex-connected planar graphs. The planarity characteristic is not the problem, since the street networks of most cities are planar too. What stands between these theorems and their application is the assumed vertex connectivity of the graphs. Recall that using Menger's theorem [27] r-vertex connectivity is equivalent to the existence of r vertex disjoint paths between every pair of vertices. The 2-vertex connectivity assumed in Theorem 2 is still moderately realistic, since there often exist two distinct routes between every two points in a city. 3-vertex connectivity already gives bigger problems. It gets problematic when we assume 4-vertex connectivity, which we do in Theorems 3-5, when we take $r \geq 4$. This is equivalent to assuming the existence of 4 distinct paths between any two points in a city. Every junction therefore needs at least 4 roads leading to it, ruling out cities with T-junctions. We can conclude that the assumption of vertex connectivity limits the number of graphs to which the theorems can be applied.

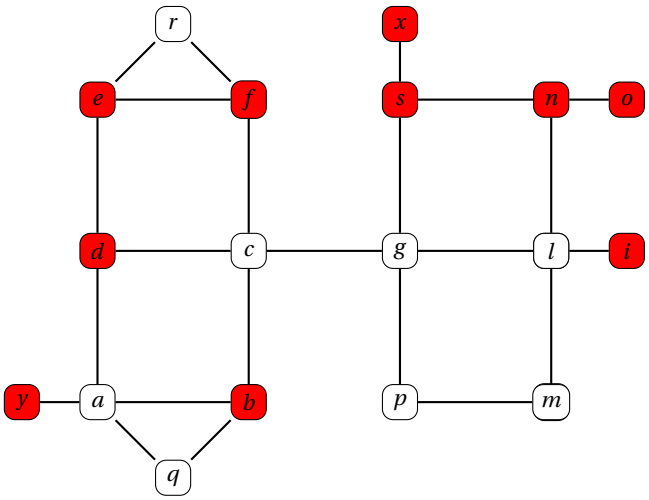
Possible solutions
What can future research do to avoid this vertex connectivity restriction and make these four theorems more applicable to debris management? Our choice for vertex connectivity as an assumption was motivated by the characteristic that 4-vertex-connected graphs have a Hamiltonian path [32]. A Hamiltonian path is a Caterpillar path and graphs with a Caterpillar path are arbitrarily edge divisible. Instead of a Hamiltonian path, one could also search for an Eulerian path. A path that visits every edge exactly once. By the same logic as in Case 4 of Lemma 3 it can be seen that graphs with Eulerian paths have the required edge divisibility too. Since graphs where every node has an even degree have such a path, we can conclude that these graphs are suitable for our theorems too. We could replace the requirement of vertex connectivity with graphs with only even degree nodes. However, this method still rules out cities with T-junctions, since T-junctions have degree 3.

There is another possibility to work around vertex connectivity. We can remove edges of the graph $G = (V, E)$ to obtain a new graph G'. As long as G' is connected, then the existence of an Eulerian path in G', is a sufficient condition for a Caterpillar path in G. This is true because an Eulerian path in G' visits every vertex in G' and, since we only removed edges, G' has the same vertex set as G. Therefore, the Eulerian path also visits all vertices in G. Hence, all edges in E are either in this Eulerian path or connected to a vertex that is in the path. The Eulerian path in G' is therefore a Caterpillar path in G. We are left with one question: how to remove edges of G such that the remaining graph G' is connected and has an Eulerian path?

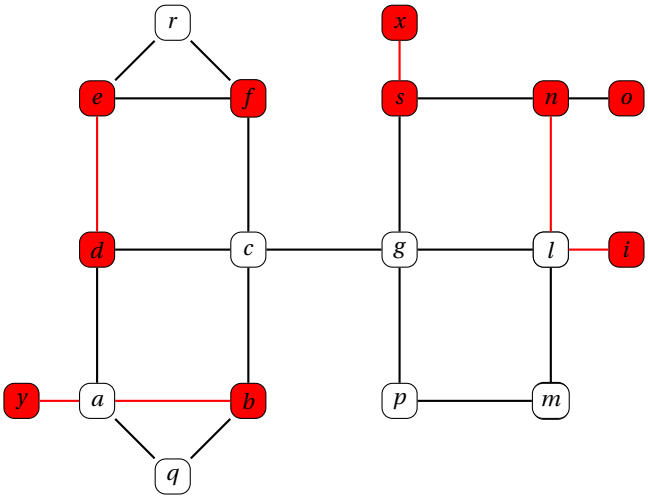
We need to remove edges in such a way that every node in the remaining graph (except for two nodes) have an even degree. One approach could be to find the set V_{odd} of all odd degree nodes in G. We then create pairs of two odd degree nodes, such that every node in V_{odd} is in exactly one pair. This is always possible, because there is an even of number of odd degree nodes in undirected connected graphs. Let us call the collection of these pairs of odd degree nodes Q. For all pairs $q = (q_1, q_2) \in Q$, except for one pair, we find a path P_q connecting q_1 and q_2 such that i) all nodes in P_q except for q_1 and q_2 are of even degree and ii) all paths P_q are edge-disjoint. There is one pair of odd degree vertices in Q that we do not connect by a path. We call the combination of the pairs Q and the paths P_q an "odd pair path set".

We now remove in the original graph G every edge that occurs in P_q for some q. In the remaining graph, we remove every vertex that has degree zero. We denote the graph we end up with by G'. Now if G' is connected and contains for every edge e from the original graph G at least one of its endpoints, we call the odd pair path set "suitable". When an odd pair path set is suitable, G' has an Eulerian path and G has a Caterpillar path.

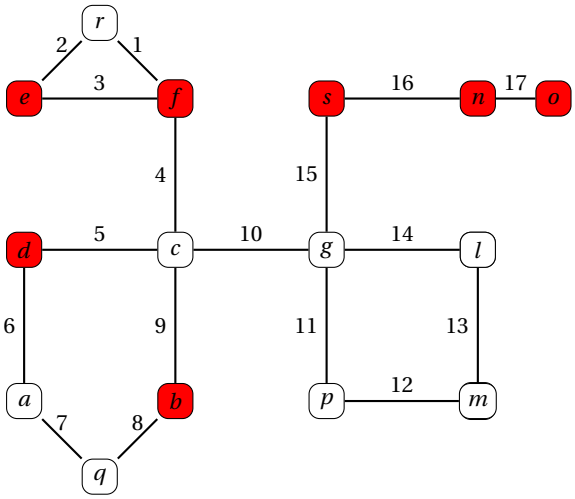
Why is this true? For some pair q of odd nodes, let v be an interior node in the path P_q. By assumption, it has even degree k. After we removed the edges in path P, the vertex v has degree $k-2 > 0$. Hence v still has an even degree. However, the degree of the starting and ending nodes of P_q, vertices q_1 and q_2, reduces by 1 and becomes even. This is true for every path P_q. Hence G' only has even nodes, except for the pair in Q that we did not connect. Let us denote the vertices in this pair by v_{start}, v_{end}. Since G' is connected , it has an Eulerian path between v_{start}, v_{end}. By assumption, every edge in G has an endpoint in this Eulerian path. Hence the Eulerian path forms a Caterpillar path in G. In Figure 9.1 it is shown how to find a Caterpillar path with this method.



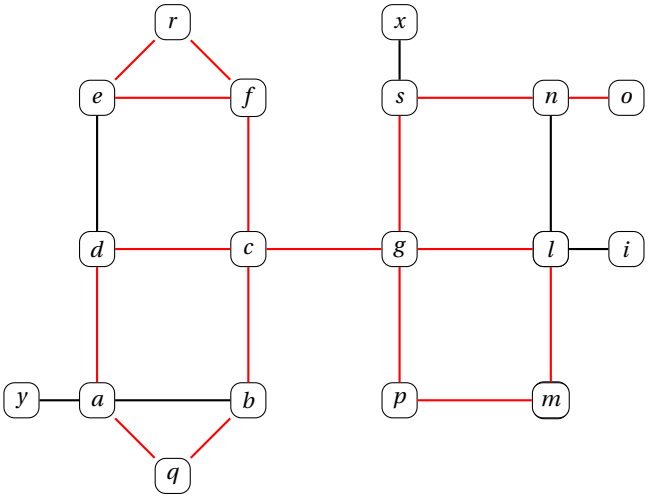
(a) A graph for which we need to find a Caterpillar path. All odd degree nodes are red.



(b) We find a suitable odd pair path set by connecting pairs q of odd nodes by a path P_q. All paths P_q are in red. Note that we did not connect the two odd nodes f and o.



(c) We remove all edges that appear in P_q for some pair q. The remaining graph G' has a Eulerian path starting in node f and ending in node o, denoted by the edge labeling.



(d) The Eulerian path in G' is a Caterpillar path in G. The Caterpillar path is denoted in red.

Figure 9.1: An example graph for which a Caterpillar path is found using the odd pair path method. With this Caterpillar path and using the method from Figure 4.4 we can now partition the edge set of this graph into connected subsets.

In Figure 9.1b we find a suitable odd pair path set. The existence of this set is sufficient for the existence of a Caterpillar path and hence, for arbitrary edge divisibility. In other words, the existence of a suitable odd pair path set would lead to an enormous improvement of the applicability of Theorems 2-5. Can we always find such a set? No, not always. In Figure 9.2 a graph is shown without an odd pair path set. Notice that for every pair of odd vertices in this graph, removing the edges of the path connecting them results in a disconnected graph.

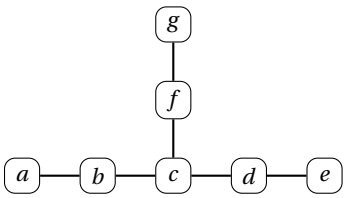


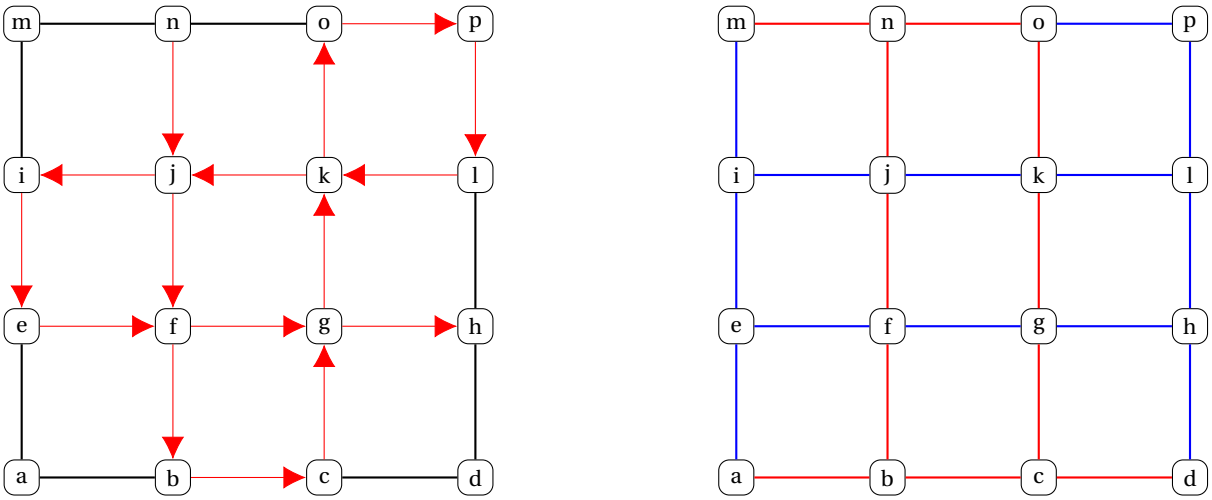
Figure 9.2: An example graph that does not have a suitable odd pair path set and therefore also no Caterpillar path.

The graph in Figure 9.2 has some nodes of degree 2. In a city these nodes do not exist, since every junction has at least 3 streets joining. Moreover, the graph in Figure 9.2 is not 2-vertex-connected. We did not manage to find an example of a 2-vertex-connected planar graph with minimal degree 3 that had no suitable odd pair path set. However, we could not prove that a suitable odd pair path set always exists in these graphs either. A proof that shows which conditions are sufficient for the existence of a suitable odd pair path set would be of great value.

9.3. Partition shape

Shortcomings

In Theorems 2-5 we show that we can find solutions that partition the edge set into r connected subsets $E_1, E_2, \ldots, E_r$, where r denotes the number of agents. In this way, every agent gets assigned exactly one connected subset of the edges. When applied to the debris management problem, this means that every contractor is assigned one connected part of the city. But connectedness is not the only desirable characteristic of a subset. We prefer to have subsets that have a compact shape, like a circle or square, rather than an awkward shape. The method using the Caterpillar path described in Lemma 3 does not necessarily give this result. In Figure 9.3 we see how a poorly chosen Caterpillar path can result in a partition that, albeit connected, is still undesirable.



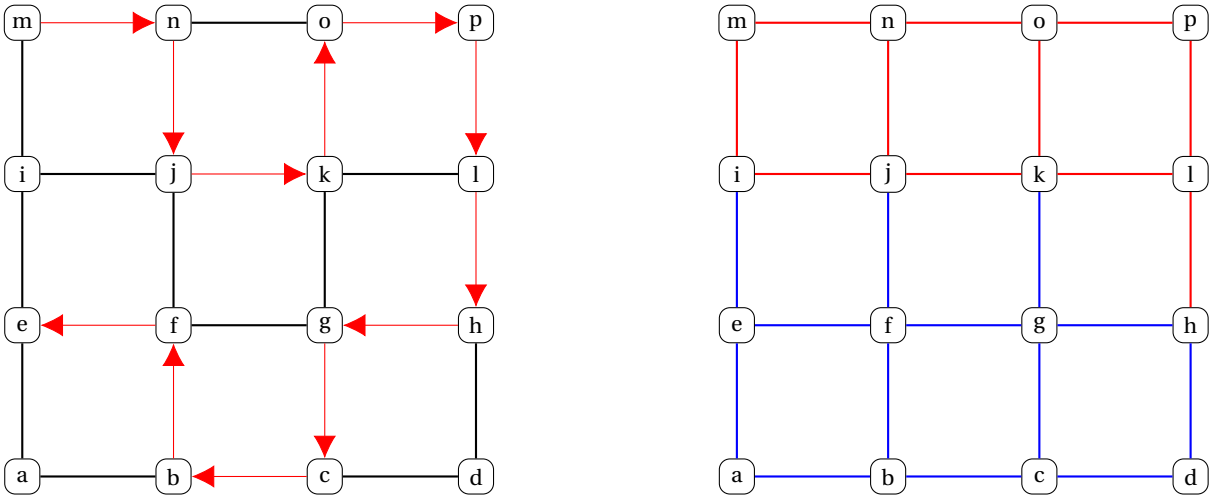
(a) An example graph with a Caterpillar path

(b) The resulting edge partition

Figure 9.3: An example graph with a poorly chosen Caterpillar graph can result in a partition that is connected, but awkwardly shaped.

Possible solutions

A solution to this particular problem is to use a different Caterpillar path. One that does result in compact subsets. In Figure 9.4 another Caterpillar path is given to the same graph as in Figure 9.3, resulting in subsets with better shapes.



(a) An example graph with a Caterpillar path

(b) The resulting edge partition

Figure 9.4: The same graph as Figure 9.3, but with a Caterpillar graph that does result in a good partition.

But finding a better Caterpillar path is hard for graphs the size of cities. An approach could be to randomly produce many different Caterpillar paths and use the one that gives the "nicest" partition. Another approach could be to write an algorithm that grows the Caterpillar path edge by edge. When there is more than one edge to add, the algorithm could pick the edge that is most suitable to add according to a metric.

9.4. Matching algorithm

Shortcomings

In our final UM algorithm (Algorithm 9) we run two other algorithms: the combination version of the starting UM algorithm and the combination version of the ending UM algorithm. To run the former algorithm we compute a matching M. In the algorithm we then allow all edges in this matching to change. All other edges are forced to stay with their current agent. Clearly, the success of the algorithm depends on this matching. For simplification reasons we computed this matching greedily using Algorithm 2, while Edmonds' blossom algorithm [6] provides a polynomial time method to find the minimum cost matching. The choice of leaving out this Edmonds' method might have resulted in a non-optimal matching and hence, in a less effective algorithm.

Possible solutions

The solution is simple. Instead of Algorithm 2 one could implement Edmonds' blossom algorithm. It would be worthwhile to observe the obtained improvement in the efficiency of the final UM algorithm.

9.5. Instance variety

Shortcomings

In Chapter 6 we describe the instances on which we test our final UM algorithm (Algorithm 9). We run instances on nine different American cities, with 5 and 10 contractors and for eight weight settings. With these choices we captured a variety of possibilities that can occur in real-life. However, we still ignored many other possible instances. The variety of instances treated in this thesis was lacking in four main areas: i) the amount of contractors, ii) the weight settings, iii) the cities and iv) the values of α.

The algorithm was not tested for enough different amounts of contractors. We only considered the case of 5 and the case of 10 contractors. In real cases the amount of contractors can be anything bigger than two.

The algorithm was not tested for enough different weight settings. This thesis only considered disasters that have the same impact around a city. We showed in Chapter 6 that this holds for earthquakes and hurricanes. However, floods, wildfires, landslides and tornadoes do typically cause certain parts of the city to be more severely struck than other parts. By limiting ourselves to location independent weight settings, we have no information on the applicability of our algorithm on these kinds of disasters.

The algorithm was not tested for enough different cities. Only American cities were considered. We therefore have no information on the applicability of our algorithm on other cities. Due to the robustness of our algorithm we expect the algorithm to work on all cities worldwide. Further research should show whether this is true.

The algorithm was tested for three values of α: 0.8, 0.7 and 0.6. It is valuable to know how the algorithm performs for more values of α

Possible solutions
It is valuable to know how the final UM algorithm (Algorithm 9) would perform on the instances left out in this thesis. Future research can test the performance of our algorithm for different amounts of contractors, location dependent weight settings, cities outside of the United States and more values of α. It is interesting to see what the effect is on the recommendations discussed in Chapter 7.

In Chapter 7 we saw that in the case of 10 contractors the algorithm could not produce solutions with a small number of zones. Even for the lowest value of α, 0.6, the algorithm partitioned the graph in at least 40 zones. What would happen when we decrease the value of α even further? Figure 9.5 shows that in this case good solutions can be found.

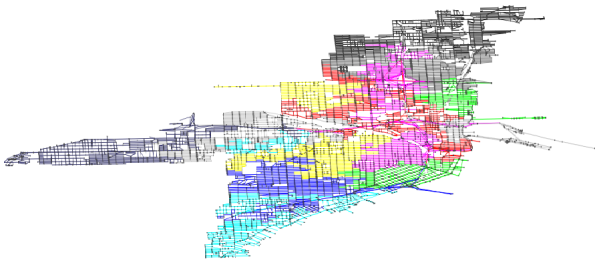


Figure 9.5: A solution of the final UM algorithm on Miami, Florida using weight setting 1, 10 contractors and $\alpha = 0.5$

Because of the robustness of the algorithm, we hypothesize that when $\alpha = 0.5$, the algorithm can find solutions with a small number of zones for other cities and weight settings too.

Bibliography

[1] T. Allen, D. Wald, and C. Worden. Intensity attenuation for active crustal regions. *Journal of Seismology*, 16(3):pp 409 433, 2012. doi: https://link.springer.com/article/10.1007/s10950-012-9278-7.

[2] G. Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017. doi: https://geoffboeing.com/publications/osmnx-complex-street-networks/.

[3] F. Bourse, M. Lelarge, and M. Vojnovic. Balanced graph edge partition. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1456–1465, 2014. doi: https://dl.acm.org/citation.cfm?id=2623660.

[4] B. Bozkaya, E. Erkut, and G. Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operations Research*, 144(1):12–26, 2003. doi: https://www.sciencedirect.com/science/article/abs/pii/S0377221701003800.

[5] E. Edis, C. Oguz, and I. Ozkarahan. Parallel machine scheduling with additional resources: Notation,classification, models and solution methods. *European Journal of Operational Research*, 230(3): pp 449–463, 2013. doi: https://www.sciencedirect.com/science/article/abs/pii/S0377221713001914.

[6] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:pp 449–467, 1965.

[7] R. Edwards. How does the damage from hurricanes compare to that of tornadoes? doi: https://www.americangeosciences.org/critical-issues/faq/how-does-damage-hurricanes-compare-tornadoes.

[8] T. Ellis. Detroit, indianapolis and buffalo among the least disaster-prone and most affordable places to live. 2019. doi: https://www.redfin.com/blog/natural-disaster-hazard-score-by-metro-area/.

[9] M. Garey and D. Johnson. Computers and intractability:a guide to the theory of np-completeness. 1990. doi: https://dl.acm.org/citation.cfm?id=574848.

[10] R. Garfunkel and G. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8):pp B495–B508, 1970. doi: https://www.jstor.org/stable/2628656?seq=1.

[11] B. Gavish and H. Pirkul. Algorithms for the multi-resource generalized, assignment problem. *Management Science*, 37(6):695–713, 1991. doi: https://www.jstor.org/stable/2632526?seq=1.

[12] O. Goldschmidt, S. D. Hochbaum, A. Levin, and E. V. Olinick. The sonet edge-partition problem. *Networks*, 41(1), 2003. doi: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.10054.

[13] R. Graham, E.Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling graham. *Annals of Discrete Mathematics*, 5:pp 287–326, 1979. doi: https://www.sciencedirect.com/science/article/abs/pii/S016750600870356X.

[14] S. Graham and H. Riebeek. Hurricanes: The greatest storms on earth. 2006. doi: https://earthobservatory.nasa.gov/features/Hurricanes.

[15] A. Guerrieri and A. Montresor. Distributed edge partitioning for graph processing. *CoRR*, 2014. doi: http://arxiv.org/abs/1403.6270.

[16] E. Győri. On division of graphs to connected subgraphs. *Combinatorics (Proc. Fifth Hungarian Combinatorial Coll., 1976., Keszthely)*, pages pp 485–494, 1976.

[17] W. Hager, D. Phan, and H. Zhang. An exact algorithm for graph partitioning. *Mathematical Programming*, 137(1-2):pp 531–556, 2013. doi: https://link.springer.com/article/10.1007/s10107-011-0503-x.

[18] O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):pp 280–289, 1977. doi: https://dl.acm.org/citation.cfm?id=322011.

[19] R. Iyer and J. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. *CoRR*, 2013. doi: https://arxiv.org/abs/1311.2106.

[20] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. *IEEE*, 1998. doi: https://ieeexplore.ieee.org/document/1437315.

[21] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2), 1970. doi: https://ieeexplore.ieee.org/abstract/document/6771089.

[22] J. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1-3):pp 259–271, 1990. doi: https://link.springer.com/article/10.1007/BF01585745.

[23] L. Lovász. A homology theory for spanning trees of a graph. *Math. Acad. Sci. Hungaricae*, 30:pp 241–251, 1977. doi: https://link.springer.com/article/10.1007\%2FBF01896190.

[24] L. Luther. Disaster debris removal after hurricane katrina: Status and associated issues. 2008. doi: https://fas.org/sgp/crs/misc/RL33477.pdf.

[25] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):pp 1, 1959. doi: http://www.columbia.edu/~cs2035/courses/ieor6400.F07/mcn1.pdf.

[26] A. Mehrotra, E. Johnson, and G. Nemhauser. An optimization based heuristic for political districting. *Mangement Science*, 44(8):1100–1114, 1998. doi: https://www.jstor.org/stable/2634689?seq=1.

[27] K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10(1):pp 96–115, 1927. doi: https://eudml.org/doc/211191.

[28] H. Ritchie and M. Roser. Natural disasters. 2014. doi: https://ourworldindata.org/natural-disasters.

[29] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):pp 555–565, 1976. doi: https://dl.acm.org/citation.cfm?id=321975.

[30] H. Stern. Minimizing makespan for independent jobs on nonidentical parallel machines: An optimal procedure. 1975. doi: https://books.google.nl/books/about/Minimizing_Makespan_for_Independent_Jobs.html?id=5erucQAACAAJ&redir_esc=y.

[31] K. Stilp, P. Keskinocak, M. Celik, and O. Ergun. Area partitioning for debris collection. *Unpublished*, 2015.

[32] W. Tutte. A theorem on planar graphs. *Transactions of the American Mathematical Society*, 82(1):pp 99–116, 1956. doi: https://www.jstor.org/stable/1992980?seq=1.

[33] Unknown. Gurobi optimization. doi: https://www.gurobi.com/.

[34] Unknown. Debris management guide. page 43, 2007. doi: https://www.fema.gov/pdf/government/grant/pa/demagde.pdf.

[35] D. Williamson and D. Shmoys. The design of approximation algorithms. pages pp 282–285, 2010. doi: http://www.designofapproxalgs.com/book.pdf.

[36] M. Yagiura, S. Iwasakib, T. Ibarakic, and F. Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discrete Optimization*, 1:87–98, 2004. doi: https://core.ac.uk/download/pdf/82288788.pdf.