Universiteit Utrecht

Master thesis (ICA-3238687)

Separating a polygonal environment into a multi-layered environment

Supervisors:
Dr. ir. J.M. van den Akker
Dr. R.J. Geraerts
Dr. J.A. Hoogeveen

Author:
A. Hillebrand

Date:
November 28, 2012

Abstract

Path planning is the field of computer science devoted to finding efficient and/or realistic methods used to navigate characters through virtual environments. The Explicit Corridor Map [21] is a navigation mesh which has recently been extended to scenes that consist of multiple 2D layers which are connected by so called transfers [42]. Obtaining such a layered 2D environment from a 3D polygonal environment has been studied by Saaltink [40]. Saaltink employed a brute-force algorithm, as well as a graph-based algorithm.

In this study we continue searching for other algorithms employing a graph encoding of the 3D polygonal environment. In this graph encoding each polygon is represented by a vertex. Whenever two polygons are next to each other, an edge connecting the corresponding vertices is added to the graph. Lastly, a special kind of edge called an overlap is added between to vertices in the graph whenever the corresponding polygons overlap when they are projected on the xz-plane. We show that finding a multi-layered environment using this graph encoding is an NP-Hard problem by a reduction from `3-SAT`. Methods are described that can significantly reduce the size of the graph encoding of the 3D polygonal environment. Furthermore, we have implemented a range of different heuristic methods. The first heuristic method employs shortest path algorithms to quickly search and cut all paths connecting overlapping polygons. The second heuristic method clusters polygons based on height information. In addition to these two methods, different local search algorithms are implemented, as well as a genetic algorithm. Since none of these methods are guaranteed to find an optimal solution, we have also implemented an integer linear programming that employs Branch & Price. The results of all these different methods are listed and evaluated.

For these tests, nine environments were used. The height-based clustering and an implementation of local search outperform all other methods, both in quality of the solution as well as in execution time.

Keywords: Multi-layered environment; Explicit Corridor Map; Branch & Price; Local search; Genetic algorithm; `MULTICUT`; Graphs

Contents

Chapter 1

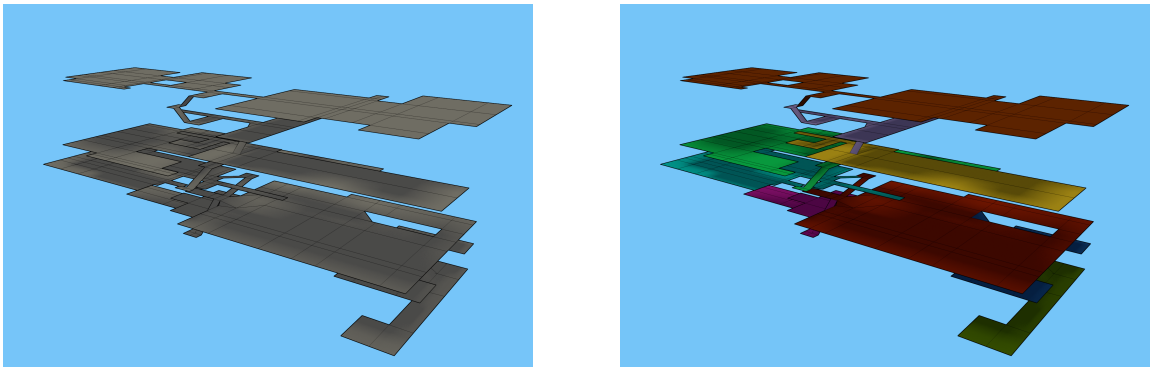
Introduction

A lot of computer games and simulations take place in a virtual world, built up off a set of polygons. Such a set of polygons is called a polygonal environment **P**. To make the worlds used for computer games more attractive and dynamic, large crowds of virtual characters are often added. In simulations large crowds of virtual characters can be used to help determine the safety of an environment. A great example of such research is carried out by the 'Fire Safety Engineering Group' at the University of Greenwich, which uses simulations to determine the fire safety of different buildings. To allow these characters to navigate the virtual landscape, a so-called navigation mesh is constructed. In the past, creating these navigation meshes was often done by hand. However, with the ever increasing size and complexity of the current virtual worlds, manual construction of these navigation meshes is no longer feasible. Creating a navigation mesh by hand would simply take too much time and would be sensitive to errors.

Therefore, several algorithms were designed to automate this process. In 'Planning Algorithms' by LaValle, several methods to generate data structures are described [32]. The Reduced Visibility Graph can find shortest paths in a 2-dimensional space and Vertical Cell Decomposition can help navigate in n-dimensional environments. However, both methods generate unnatural paths, because the virtual character walks close to walls and makes very sharp turns. Furthermore, both algorithms cannot handle dynamic changes to the environment, like other moving characters. The reason for this is that the algorithms that construct the paths on the basis of these navigation meshes do not take any dynamic changes into account.

The Explicit Corridor Map (ECM) by Geraerts and Saaltink [43] resolves the aforementioned problems of unnatural paths and single character motion planning. However, the ECM can only handle a set of 2-dimensional layers, where inside each individual layer no polygons may overlap. A decomposition of a 3-dimensional virtual environment into such a set of 2-dimensional layers is known as a Multi-layered environment (MLE) [42]. An MLE is characterized by the set of layers, as well as the points where two different layers connect, known as transfers. When an MLE is given, the construction time for the ECM is $O(kn \log n)$, where n is the number of polygons in the MLE and k the number of transfers between the different layers. The MLE as described before is used by Saaltink [40] and Van Toll [42]. Saaltink developed multiple algorithms to generate an MLE from an environment **P**, whereas Van Toll assumes that an MLE is given as input for his algorithm.

Since both k and n are important factors in the construction time of the ECM, it is prudent that the MLE used has a small number of transfers as well as layers. The goal of



(a) (b)

Figure 1.1: In (a) an example of a polygonal environment is given. In (b) this same environment is decomposed into nine layers and uses eight transfers. The number of transfers and layers for this environments is minimal.

this thesis is finding such an MLE from a 3-dimensional polygonal environment. An example of a polygonal environment and a corresponding MLE is shown in Figure 1.1.

1.1 Previous work

Manu different methods to generate a layered description of **P** exist, although none of these methods generates an actual MLE. Some of these methods are described by Saaltink [40]. Most of these methods just create heightmaps of the environment and directly use this information for path planning and collision avoidance. Other methods only detect surfaces that are connected.

For that reason Saaltink created some methods to find an MLE that has a minimal number of transfers. First the environment is transformed into a graph structure known as the Polygonal Environment Graph (PEG). In this PEG, a vertex is added for each polygon in the environment, and, whenever two polygons connect, an edge between the two corresponding vertices is added. Furthermore, information is stored in the PEG about vertices that should never end up in the same layer. Using this PEG both a brute force algorithm, as well as some heuristics are described. Although the brute force method is guaranteed to find an optimal solution, it has an execution time of $O(2^{|E|}(n+k))$ in worst case scenarios and is therefore infeasible to use in many situations. Here $|E|$ is the number of edges in the PEG, n the number of layers and k the number of transfers.

The heuristic methods Saaltink describes are mainly based on Ford and Fulkerson's max-flow min-cut theorem [17]. Using a PEG, a set of possible transfers **PT** is calculated by finding the minimal cut in the PEG between any two polygons that should be in different layers. Each edge in the resulting cut sets is deemed a possible transfer. Using a simple branching strategy, a minimal set of transfers from **PT** that corresponds to a valid MLE is found.

1.2 Goal of this thesis

Although Saaltink's algorithms seem to have great results, there is no bound on how close the resulting solutions are to the optimal solution. Furthermore, there is no theoretical basis for finding the MLE using the above mentioned graph representation. Therefore, the first goal for this thesis project will be to determine if there can exist a polynomial time algorithm for solving this problem. If this is not the case, attempts will be made to find good heuristics for finding an MLE, as well as an algorithm that should be able to find the optimal solution, despite the fact that this will take a very long time. Furthermore, attempts will be made at finding possible ways to reduce the size of the graph representation of the polygonal environment. This is done in the hopes that on the new and smaller representation of the problems, the algorithms will run significantly faster.

Afterwards, an experimental evaluation of the different algorithms will be made to determine what algorithm yields the best results when taking in consideration the quality of the solution as well as the time needed to find this solution.

1.3 Thesis outline

In Chapter 2, a formal definition of the problem will be given, as well as definitions of polygons, polygonal environments and an MLE. Furthermore, a graph structure containing all relevant information of a polygonal environment $\mathbf{P}$ will be defined. In Chapter 3 it will be shown that there exists no polynomial time algorithm that can find an optimal MLE. Finding an optimal MLE turns out to be a NP-Hard problem when we want to find the minimal set of transfers for a given polygonal environment.

Chapter 4 contains the basis for a divide-and-conquer strategy, as well algorithms that can be used to reduce the size of the graph, while an optimal solution can still be found. In Chapter 5 a local search and evolutionary algorithm will be described, as well as two greedy heuristic methods. One method is based on cutting all shortest paths between overlapping vertices, and the other one is based on clustering polygons together based on height information. A linear programming formulation of the problem is given in Chapter 6. The experiments conducted will be described in Chapter 7 and the results will be discussed in Chapter 8. In Chapter 9, we conclude that on the tested environments the height based heuristic outperforms all other algorithms, closely followed by local search. Furthermore, running the tested algorithms on the reduced graph representations does decrease the execution time. Unfortunately however, reducing the graphs takes more time than that is gained by using these reduced graphs. Furthermore, a list of still open problems will be given.

Chapter 2

Problem definition

In this chapter a formal definition of the problem outlined in Chapter 1 will be given. This chapter considers an Euclidean environment $\mathbb{R}^3$. The notation for a point or location in this environment is $l = (l_x, l_y, l_z)$, where the l_y-component of a point gives the vertical position of this point. Besides points, polygons can be defined in this environment. Such an environment containing polygons will be called a polygonal environment. The notation used for a polygonal environment is $\mathbf{P}$. In the upcoming sections, some definitions will be given concerning the environment $\mathbf{P}$.

First, polygons as they are used in the remainder of this thesis will be defined. Next, in Section 2.2 a definition of polygonal environments and some properties of these environments will be introduced. In the last section, definitions of a multi-layered environment and transfers will be given. Furthermore, a graph structure will be defined that contains all the information from a polygonal environment needed to create a multi-layered environment which has a minimal number of transfers.

2.1 Polygons

A polygon P is defined by a list of coplanar vertices $\mathbf{p} = \{p_0, \ldots, p_{n-1}\}$ where $n \geq 3$. Each $p_i \in \mathbf{p}$ is an unique corner of polygon P and the points in $\mathbf{p}$ are ordered in a clockwise or counter-clockwise order. Since all vertices in $\mathbf{p}$ lie on one and the same plane, the corresponding plane can be found using three points from the list $\mathbf{p}$. Using these points, the normal vector of the plane $\vec{n}$ can be determined using the equation $\vec{n} = (p_i - p_0) \times (p_j - p_0)$. Here, $i \neq j$ and $1 \leq i, j \leq n-1$. Furthermore, p_0, p_i and p_j should not lie on the same line. If p_0, p_i and p_j are located on the same line, the resulting cross product would be the null vector.

Using the list of coplanar vertices $\mathbf{p}$, n lines can be defined between the consecutive points of $\mathbf{p}$. The points on these lines are given by the equation $e_i(t) = t(p_{(i+1)\%n} - p_i) + p_i$, where $i = \{0, \ldots, n-1\}$ and % is the modulo operation. The line segments $e_i(t), i \in \{0, \ldots, n-1\}, 0 < t < 1$ are called the edges connecting vertices p_i and $p_{(i+1)\%n}$. It is assumed that none of the n edges belonging to polygon P intersect, i.e. P is a simple polygon. Using these n lines, a polygon P is defined as the open set of points that lie within the bounded area defined by these n lines and on the plane defined by p_0 and $\vec{n}$ as described in Figure 2.1. The reason for the definition of polygons as an open set is that otherwise neighbouring polygons (polygons that share an edge) would also share the points on that edge. This would in turn cause the definition of connecting and overlapping polygons later on to be cumbersome.

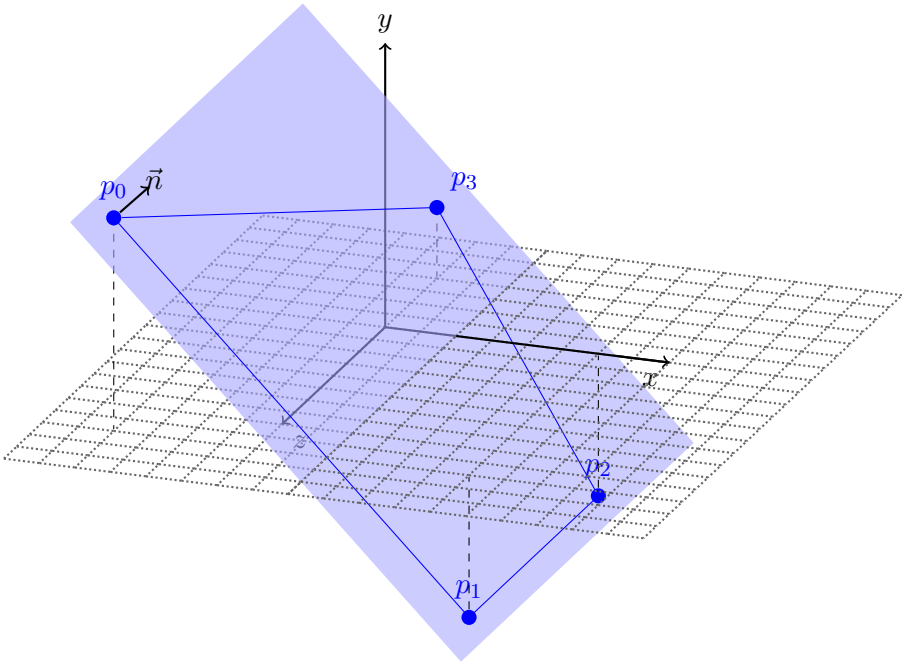


Figure 2.1: The polygon is defined by the vertices $\mathbf{p} = \{p_0, p_1, p_2, p_3\}$. All vertices lie on the plane defined by $(p_0, \vec{n})$. A part of this plane is shaded blue.

A direct consequence of the definition of polygons as open sets is that the coplanar vertices $\mathbf{p}$ and the points on the edges e_i, $i \in \{0, \ldots, |\mathbf{p} - 1|\}$ are not points on the polygon. These points are only used to define the polygon.

Two polygons $P = (\vec{n}_p, p_0)$ and $Q = (\vec{n}_q, q_0)$ might intersect when the normal vectors $\vec{n}_p$ and $\vec{n}_q$ are not parallel. When this is the case, it is possible to find a line along which the corresponding planes intersect. The points on this line are given by $e_{int}(t) = l + (\vec{n}_p \times \vec{n}_q)$. The location l has to be located on both planes, and can be found by determining a point that satisfies both $\vec{n}_p \cdot l = \vec{n}_p \cdot p_0$ and $\vec{n}_q \cdot l = \vec{n}_q \cdot q_0$. Using the line defined by $e_{int}(t)$ and the edges of both polygons, the two polygons intersect if $e_{int}(t)$ intersects edges of both polygons but is not equal to the actual edges $e_i(t)$ of the polygons.

Furthermore, two polygons can be *connected* or *overlapping*. A polygon P is defined to be connected to a polygon Q if they share an edge. When P is connected to Q, this will be denoted by $\rightleftharpoons$.

Definition 1. *Polygons P and Q are* **connected** *(denoted $P \rightleftharpoons Q$) when P and Q share at least one entire edge.*

Two polygons P and Q can also be placed above each other. When this happens, it is possible for two polygons to have overlapping points when they are projected onto the xz-plane (ground-plane): P and Q overlap when a (sub)set of the points of P is contained within Q when both P and Q are projected on the ground plane.

Definition 2. *Polygons P and Q* **overlap** *(denoted $P \Updownarrow Q$ or $Q \Updownarrow P$) if there exists at least one point $l = (l_x, l_y, l_z)$ in polygon P and at least one point $m = (m_x, m_y, m_z)$ in polygon Q with $(l_x, l_z) = (m_x, m_z)$.*

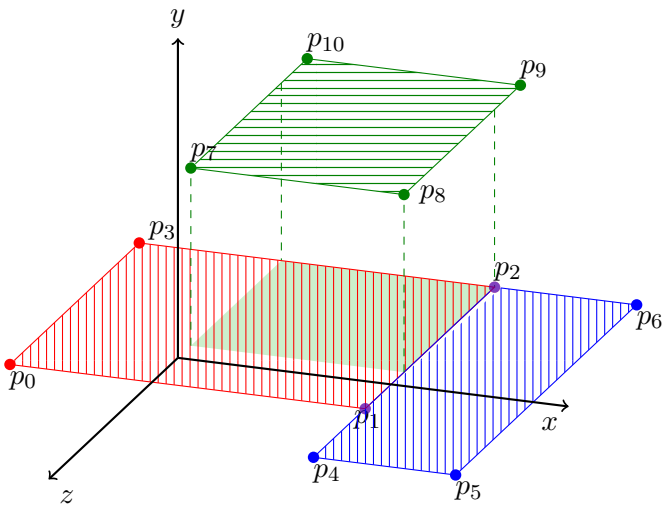


Figure 2.2: Three polygons $P_{red} = \{p_0, p_1, p_2, p_3\}$, $P_{blue} = \{p_1, p_4, p_5, p_6, p_2\}$ and $P_{green} = \{p_7, p_8, p_9, p_{10}\}$. The following statements hold: $p_{red} \rightleftharpoons p_{blue}$, $p_{green} \Updownarrow p_{red}$, $p_{green} \nUpdownarrow p_{blue}$

These relations between polygons are shown in Figure 2.2. It is important to note that the green polygon P_{green} does not overlap the blue polygon P_{blue}, although the points p_8 and p_1 and the points p_9 and p_2 are placed directly above each other. Since polygons are defined as open sets, the corners and the points on the edges of each polygon are not actually part of the polygon itself.

2.2 Polygonal environments

A polygonal environment (denoted $\mathbf{P}$) is defined on $\mathbb{R}^3$. An unrestricted polygonal environment can contain any number of polygons with all possible orientations and locations. However, the environment under consideration here has some additional restrictions. First of all, it is assumed that all polygons $P \in \mathbf{P}$ are convex. It is also assumed that there are no polygons that intersect. Both these requirements can be fulfilled by pre-processing $\mathbf{P}$. Transforming concave polygons into convex polygons can be done by using triangulation or tessellation, and splitting intersecting polygons into multiple polygons can be done using one of the many clipping techniques. Examples of these techniques were implemented by Fournier [18] and Cyrus [13].

Furthermore, it is assumed that $\mathbf{P}$ is both **walkable** and **realistic**. An environment $\mathbf{P}$ is considered walkable if for every polygon $P \in \mathbf{P}$, a virtual entity can be located on all points that are part of this polygon.

Definition 3. *A polygonal environment* $\mathbf{P}$ *is considered walkable if* $\forall P \in \mathbf{P} : \forall l \in P : l$ *is a valid location for any virtual character under consideration.*

This means that polygons cannot be positioned vertically. If this were the case, a virtual character would have to stand on an area of zero size. Furthermore, it would also mean that it is not possible for $P, Q \in \mathbf{P}$ to be both connected and overlapping. Consider Figure 2.3. In this figure it is not possible to add the red area as a new polygon P. When this polygon

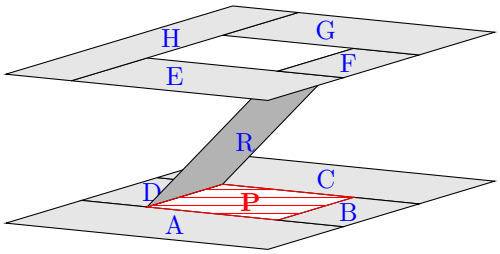


Figure 2.3: A (weakly) walkable polygonal environment **P**. When the red polygon P is added, the environment is no longer walkable; it remains weakly walkable though.

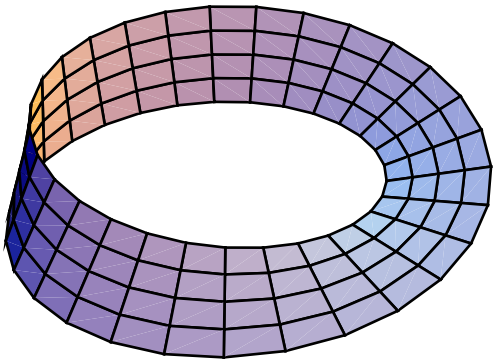


Figure 2.4: An example of a Möbius strip[44]

P would be added, it is connected to all five surrounding polygons A, B, C, D and R. Since along the edge connecting P and R only a virtual character of size zero can be located, polygon P is not walkable and therefore the environment would no longer be walkable.

Since the goal is to find valid input for the Explicit Corridor Map (ECM) [43] as explained in Chapter 1, the concept of walkability is too strict to be used in this case. The ECM is developed to handle virtual characters of different sizes. For this reason, we will use the concept of weakly walkable:

Definition 4. *A polygonal environment* **P** *is considered weakly walkable if* $\forall P \in \mathbf{P} : \exists l \in P :$ *l is a valid location for a virtual character of size* 0.

In most cases this will mean that situations as described in Figure 2.3 will be allowed as input for the algorithms described in the remaining chapters. The ECM will ensure that virtual characters will only be positioned on points of polygons where they can be positioned.

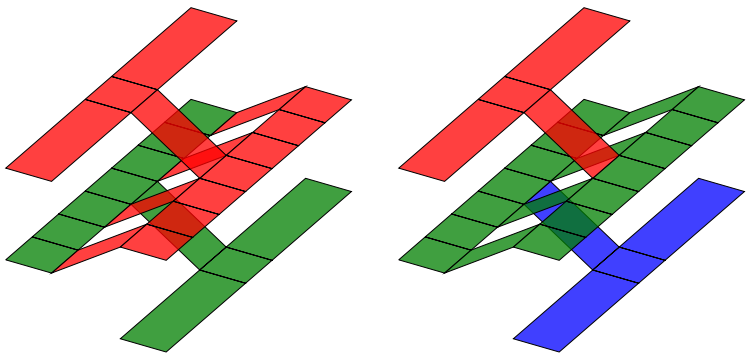
Besides (weakly) walkable, an environment **P** must also be realistic. An environment is realistic when it can be reconstructed in the real world and the walkable areas of the polygons are walkable in the real world. This can be done by checking the angle between the normal of the polygons and the vertical vector $(0, 1, 0)$. When this is more than a certain threshold (say $40°$), the surface of a polygon is not walkable in realistic situations. For instance, a polygonal environment containing a Möbius strip is not realistic. An example of the Möbius strip is given in Figure 2.4.

2.3 Multi-layered environments

Using the definitions introduced in the previous sections, a multi-layered environment $\mathbf{E}$ is defined as a collection of layers $\mathbf{L} = \{L_1, \ldots, L_n\}$ and a set of transfers $\mathbf{T}$ between the different layers. Each layer consists of a disjoint set of polygons from $\mathbf{P}$ such that $\mathbf{P} = \cup_{i=1}^{n} L_i$. Furthermore, no single polygon in L_i may overlap with another polygon in L_i ($\forall P \in L_i : \neg\exists Q \in L_i : P \Updownarrow Q$). A transfer is needed when two polygons that are connected in $\mathbf{P}$ are assigned to two different layers. Therefore, the set of transfers can be defined as $\mathbf{T} = \{(L(P,Q)|P \rightleftharpoons Q \wedge P \in L_i \wedge Q \notin L_i\}$. The function $L(x,y)$ returns the edge $e(t)$ that both the polygons x and y share. Furthermore, it is assumed that $\mathbf{P}$ forms a connected environment. This means that it is possible to travel from all polygons to all other polygons within $\mathbf{P}$. When this is not the case, the problem can easily be separated in several subproblems, solving the problem for each separate connected group of polygons contained within $\mathbf{P}$.

The challenge is to find a partition of $\mathbf{P}$ into $\mathbf{E}$ such that both the number of layers ($|\mathbf{L}|$) and the number of transfers ($|\mathbf{T}|$) is minimized. Unfortunately, it is not the case that finding the minimum number of layers also guarantees the minimal number of transfers. This is illustrated in Figure 2.5. When we have a multi-layered environment that is minimal with respect to the number of layers, we have a `MIN-L-MLE` and when we have a multi-layered environment that is minimal with respect to the number of transfers we have a `MIN-T-MLE`. Suppose we are given a `MIN-T-MLE`. Since $|\mathbf{T}|$ is minimal and there has to be at least one transfer between two layers, there can be at most $|\mathbf{T}| + 1$ unique layers for a connected polygonal environment. When a `MIN-L-MLE` is given, it is not possible to say anything about the upper bound of $|\mathbf{T}|$ given $|\mathbf{L}|$.

This observation is important since the results of this thesis will be used as input for the ECM [43] and the ECM's runtime is dependent on the total number of transfers. In the current implementation of the ECM, the runtime is dependent on both the number of layers and the number of transfers. However, it is theoretically possible to combine all the layer-dependent computations in one single step, therefore decreasing the need for a minimal number of layers. Because of these reasons, the main goal of this thesis will be to find `MIN-T-MLE`s.



(a) (b)

Figure 2.5: A polygonal environment decomposed into two different multi-layered environments. In (a) the polygonal environment is decomposed into two layers with four transfers. The environment in (b) exists out of three layers with only two transfers. These figures are based upon an example from Saaltink [40].

To make it easier to reason about this problem, we will introduce a graph-structure called the **P**-Graph which contains all essential information of **P** for finding a `MIN-T-MLE`.

Definition 5. *The* **P***-Graph* G *is defined on a weakly walkable and realistic polygonal environment* **P** *and consists of a set of vertices* V*, a set of overlapping vertices* O *and a set of edges* E*, where:*

- $V = \{v_P | \forall P \in \mathbf{P}\}$;
- $O = \{(v_P \Updownarrow v_Q) | \forall v_P, v_Q \in V : P \Updownarrow Q\}$;
- $E = \{(v_P \rightleftharpoons v_Q) | \forall v_P, v_Q \in V : \{P \rightleftharpoons Q\}$.

An example of such a **P**-Graph is given in Figure 2.6. Using the **P**-Graph, the problem of partitioning **P** into **E** while minimizing $|\mathbf{T}|$ is now equivalent to finding a minimal set $S \subseteq E$ such that there exists no path from v_P to v_Q through $E \backslash S$ for all $(v_P \Updownarrow v_Q) \in O$. This set of edges S is called a the `MIN-T-MLE` cut set. The problem of finding this `MIN-T-MLE` cut set is called `FIND-MIN-T-MLE`-cut.

Definition 6. *The problem* `FIND-MIN-T-MLE`*-cut is defined as finding a minimal set* $S \subseteq E$ *such that all overlapping pairs* $(v_P \Updownarrow v_Q) \in O$ *are separated in* $G = (V, E \backslash S)$*. The set* S *is known as the* `MIN-T-MLE` *cut set.*

When this set S is found, it encodes the information of the set **T**. From V and $E \backslash S$, the information for the different layers can be extracted. Each individual connected component in $(V, E \backslash S)$ forms one layer and each member of S forms a transfer. Since all connected components in $(V, E \backslash S)$ are disjoint and for each polygon in **P** one vertex was created, it can be shown that $\mathbf{P} = \cup_{i-1}^{n} L_i$ holds. Furthermore, in each resulting layer L_i, there are no overlapping polygons, since that would mean that there still exists a path from v_P to v_Q through $E \backslash S$ for some $(v_P \Updownarrow v_Q) \in O$.

Since S is supposed to be the minimal subset of E such that a decomposition of **P** into multi-layered environment **E** is possible, it is the case that S encodes **T**. If there is an edge in S that does not need to be in **T**, S is not minimal. Moreover, if it would be the case that there is a transfer $t \in \mathbf{T}$ which is not encoded in S, the two vertices that are connected by t are part of the same layer. Therefore, t cannot be a transfer.

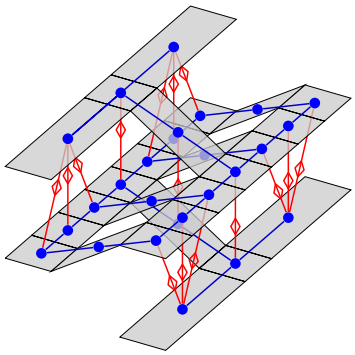


Figure 2.6: Example of a **P**-Graph belonging to the environment as seen in Figure 2.5. Each blue dot is a vertex, every blue line an edge and every red line depicts an overlap.

Chapter 3

Hardness

In this chapter, the hardness of finding a `MIN-T-MLE` in a given **P**-Graph G will be discussed. Section 3.1 shows that using max-flow min-cut is insufficient for finding a minimal decomposition of G with respect to the number of transfers.

Section 3.2 shows that the decomposition of G into a `MIN-T-MLE` can be done using the `MULTICUT` problem which is unfortunately an NP-Hard problem. Next, we will prove that finding a `MIN-T-MLE` using a **P**-Graph is an NP-Hard problem in Section 3.3. This proof is based on a paper by Dahlhaus et al. [14]. In the last section, we will take a closer look at the **P**-Graph. Since a **P**-Graph is constructed from a convex, realistic and weakly walkable polygonal environment **P**, it is not allowed to add edges or overlaps between arbitrary vertices in the corresponding **P**-Graph. This restriction on the shape of the **P**-Graph might result in bounds on the treewidth which could result in an FPT-time algorithm.

3.1 Max-flow min-cut

The max-flow min-cut theorem is a famous theorem coined by Ford and Fulkerson [17] as well as Elias, Feinstein and Shannon [15] in 1956. The max-flow min-cut theorem made it possible to find a minimal set of edges separating two vertices, often referred to as the source (s) and the sink (t), of a (directed) graph $G = (V, E)$ ($G = (V, A)$) in polynomial time. This is done by first finding the maximum flow that is possible between the vertices s and t. Afterwards, by using the max-flow min-cut theorem it is possible to determine the edges that form the min-cut separating s from t. Calculating the maximum flow can be done in $O(|V||E| + |V|^2 \log U)$ time by using an algorithm developed by Ahuja et al. [3]. Here, U is the value of the highest capacity assigned to an edge in E. Whenever the maximum flow between s and t has been found, a minimal cut can be found by performing a Depth First Search (DFS) starting from s in $O(|V| + |E|)$ time. This DFS should only traverse edges that are not saturated (i.e. the flow through these edges that was found using the max-flow algorithm should be less than the capacity of these edges). All vertices that are found using this DFS are part of the set S and all remaining vertices are part of the set T. Now the edges in the cut set are the edges (s_i, t_j) where $s_i \in S$ and $t_j \in T$.

The max-flow min-cut theorem can be applied to the `MIN-T-MLE` problem by using two different methods. The first method searches for a minimal cut for each pair of vertices $(s \Updownarrow t) \in O$. This method does not guarantee finding an optimal solution, although the solution found is valid. This method, due to Saaltink [40], is described in more detail in Section 5.1. The second method adds a so-called super source and a super sink and requires

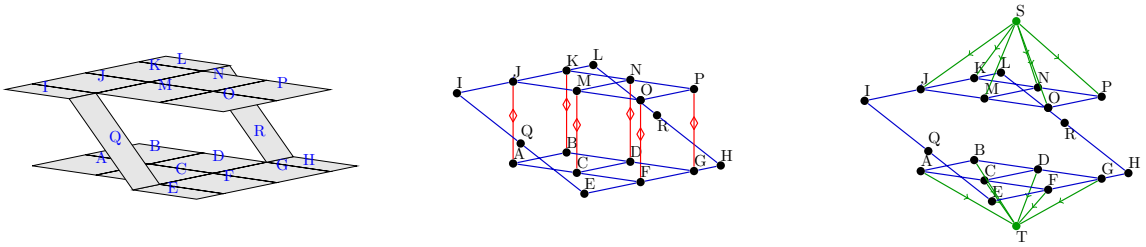
only finding one minimal cut. Unfortunately, this method can result in invalid results. We will describe below how this method can be applied to a **P**-Graph and why this method can fail.

Two simple steps are required to adapt a given **P**-Graph for use with the max-flow min-cut theorem.

1) Add a source s and a sink t to V $(V' = V \cup \{s, t\})$;

2) For each $(v \Updownarrow w) \in O$, create an arc $(s \to v)$ and an arc $(w \to t)$ with capacity 1.

These two steps can be applied in $O(|O|)$ time.



(a) (b) (c)

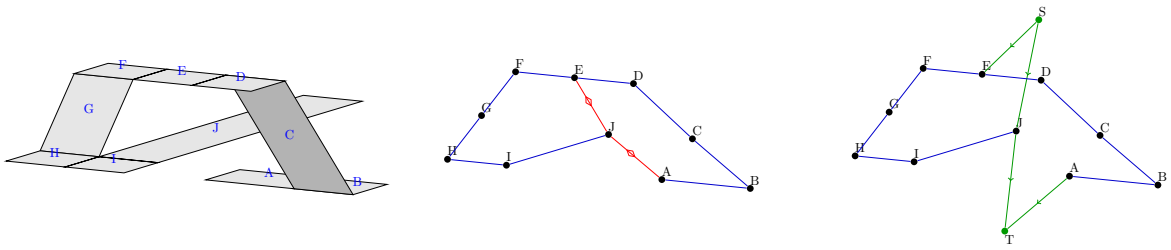
Figure 3.1: A straightforward application of the max-flow min-cut theorem to a polygonal environment. In (a), the original polygonal environment is shown. (b) shows the corresponding **P**-Graph and (c) shows the application of min-cut max-flow to this graph as described above. The min-cut found will be of size 2.

In Figure 3.1, an example of the application of the min-cut max-flow theorem is given for a given polygonal environment. The capacity of the outgoing arcs of vertex S and the incoming arcs of vertex T should be set sufficiently high to ensure that it will not be profitable to cut them. Furthermore, it ensures that the maximal flow will not be limited by the capacities of these arcs. A resulting min-cut could be the cut $\{(I \rightleftharpoons J), (R \rightleftharpoons H)\}$ of size 2.

It is easy to see that, applying min-cut max-flow to this instance will result in a feasible multi-layered environment. When one vertex overlaps another vertex, both vertices will end up in different layers. Unfortunately, applying min-cut max-flow will not always find an optimal cut set. Two problems that might occur are the introduction of more constraints and the attachment of both a source- and a sink-arc to the same vertex. Figure 3.2 is an example of a situation where both problems occur.

When applying max-flow min-cut as described above to the **P**-Graph, the separation of E from A is introduced. Furthermore, vertex J is directly attached to both the source S and the sink T. Separating S from T will result in a min-cut of size 3 (one of $(S \to J)$ and $(J \to T)$ will be cut too), while it is possible to achieve a `MIN-T-MLE`-cut of size 1.

The reason for the aforementioned problems is that in the max-flow min-cut framework of Ford and Fulkerson, there is no way to separate the individual constraints. In some environments, for example the one given in Figure 3.1, this is no problem. In this environment there are also new constraints introduced (e.g. the constraint that vertex O should be separated from $\{A, B, C, D, F, G\}$ instead of just $\{F\}$). However, these new constraints do not force a suboptimal solution to be found in this particular case.



(a) (b) (c)

Figure 3.2: A straightforward application of the max-flow min-cut theorem to a polygonal environment. In (a), the original polygonal environment is shown. (b) shows the corresponding **P**-Graph and (c) shows the application of min-cut max-flow to this graph as described above. Any valid min-cut separating S from T will also separate E from A, while this is not necessary. The min-cut found using this method equals 3, while the optimal MIN-T-MLE-cut is of size 1.

3.2 The MULTICUT problem

To resolve these issues, we turn to the multi-commodity minimal-cut problem (MULTICUT for short) as described by Schrijver [41]. In the MULTICUT-framework multiple source-sink pairs (s_i, t_i) exist, where each pair generates its own specific flow f_i. A flow belonging to such a pair can only flow from s_i to t_i, and not to another sink t_j $(j \neq i)$. The objective of MULTICUT is to find a global minimal cut, where no residual flow between any source-sink pair remains. More formally, the MULTICUT problem is defined as follows:

Definition 7. *The MULTICUT problem is defined on an undirected graph $G = (V, E)$ and a terminal pair set $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ where $s_i, t_i \in V$ for all $i : 1 \leq i \leq k$. The goal is to find a minimal set of edges $M \subseteq E$ such that each terminal pair (s_i, t_i) is disconnected in the graph $G' = (V, E \backslash M)$.*

Using MULTICUT we can find a MIN-T-MLE-cut on a **P**-Graph by modifying it in the following way:

1) For each $(v \Updownarrow w) \in O$, create a terminal pair (S_v, T_w)[1];

2) Add the arc $(S_v \rightarrow v)$ and the arc $(w \rightarrow T_w)$;

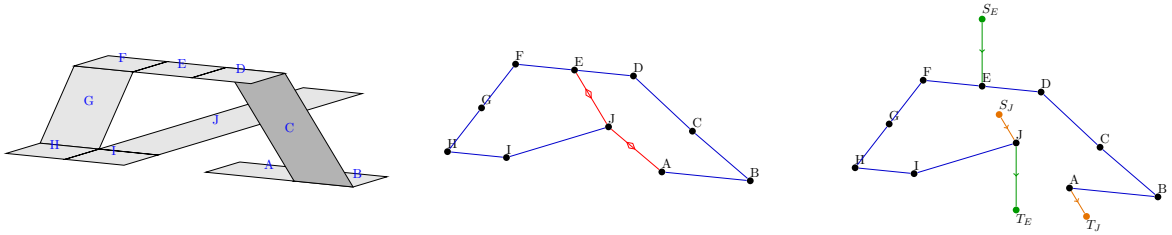
3) Remove all $(v \Updownarrow w)$ from O.

This idea is depicted in Figure 3.3. The reason why this works is that each individual constraint is assigned a specific flow. Because of this no flow can exist between sources and sinks belonging to different commodities/constraints.

When above steps are applied on a **P**-Graph, k vertices are inserted. Here k is the size of the set of terminal pairs, which is the same as size of the set $|O|$. Since for each source-sink pair one commodity is needed, we can solve the MIN-T-MLE problem by solving the MULTICUT problem with k commodities. Unfortunately, MULTICUT is proven to be NP-Hard [9] as well as

[1]It is possible to reduce the number of introduced vertices and needed flows by reusing some vertices and flows. For illustrative purposes, this is not done.

APX-Hard [11] when $k \geq 3$. When $k = 1$, MULTICUT becomes the same problem as max-flow min-cut which is solvable in polynomial time. For $k = 2$, polynomial time algorithms exist to solve MULTICUT [26, 27]. If $k \geq 3$, it can be solved in FPT-time if both the treewidth ω and the number of commodities k are given [23]. Furthermore, MULTICUT remains NP-Hard even for planar graphs with outer terminals [5] and when the input graph is a tree [20]. Since MULTICUT is such a computationally hard problem, we can not use MULTICUT to find solutions for MIN-T-MLE. Furthermore, since MIN-T-MLE looks a lot like a restricted version of MULTICUT, this suggests that MIN-T-MLE is also NP-Hard. A proof that MIN-T-MLE is indeed NP-Hard will be given in the next section.



(a) (b) (c)

Figure 3.3: An application of multi-cut to a polygonal environment. In (a), the original polygonal environment is shown, (b) shows the corresponding **P**-Graph and (c) shows how multi-cut can be applied to the **P**-Graph. Any valid multi-cut separating both terminal pairs (S_E, T_E) and (S_J, T_J) will separate E from J, and J from A by removing one of the edges (E, F), (F, G), (G, H), (H, I) or (I, J).

3.3 Proof of NP-Hardness

The proof of NP-Hardness for the MIN-T-MLE problem is based on the proof provided by Dahlhaus et al. [14]. Dahlhaus et al. prove that the MULTITERMINAL-CUT problem (MTC) is NP-Hard. Their proof is based on a polynomial time reduction of the PLANAR 3-SAT problem (P3S), which is a special case of the 3-SAT problem, to MTC. In Section 3.3.1 the problems mentioned above will be described. Section 3.3.2 will contain a sketch of the proof that MTC is NP-Hard, as given by Dahlhaus et al. [14]. Finally, in Section 3.3.3 it will be shown how the proof from Section 3.3.2 can be adapted to proof that MIN-T-MLE is NP-Hard.

3.3.1 Descriptions of 3-SAT, P3S and MTC

The 3-SAT problem is a well known NP-Complete problem [29]. In this problem, a set of n boolean variables $\mathbf{X}$ and a set of m 3-element clauses $\mathbf{C}$ are given. For each variable $\mathbf{x}_i \in \mathbf{X}$ there exist two literals, denoted x_i and $\neg x_i$. Using these literals the set $\mathbf{L}$ is defined as $\mathbf{L} = \cup_{\mathbf{x}_i \in \mathbf{X}} \{x_i, \neg x_i\}$. A clause $c_j \in \mathbf{C}$ exists out of three elements of $\mathbf{L}$. The elements of a clause c_j will be denoted $c_{j,k}$ for $k \in \{1, 2, 3\}$. Using $\mathbf{X}$ and $\mathbf{C}$, the actual problem is the question whether there exists a boolean assignment to the variables in $\mathbf{X}$ such that the formula $F = \wedge_{c_j \in \mathbf{C}} f(c_j)$ evaluates to TRUE. Here $f(c_j) = (c_{j,1} \vee c_{j,2} \vee c_{j,3})$.

P3S is a restricted version of 3-SAT and is also known to be NP-Complete [19, 33]. In P3S, we also have a set of variables $\mathbf{X}$, clauses $\mathbf{C}$ and literals $\mathbf{L}$. Furthermore, a bipartite graph

$G_{\mathbf{X},\mathbf{C}}$ is given, where for each $\mathbf{x}_i \in \mathbf{X}$ and for each $c_j \in \mathbf{C}$ a vertex is added to $G_{\mathbf{X},\mathbf{C}}$. An edge is added between $\mathbf{x}_i$ and c_j whenever one of the literals $\{x_i, \neg x_i\}$ is in c_j. The formula F for a P3S problem is restricted to formulas of which a planar graph $G_{\mathbf{X},\mathbf{C}}$ exists. To check if a graph is planar we can use graph minors in conjunction with Wagners theorem. For a definition of graph minors, take a loot at Section 3.4. According to Wagners theorem, a graph is planar if and only if it contains no K_5 or $K_{3,3}$ minors. K_5 is the complete graph on five vertices and $K_{3,3}$ is the complete bipartite graph consisting out of 3 vertices in partition one and 3 vertices in partition two. In practice, this theorem is difficult to use, however algorithms exist that can check in $O(|V|)$ time if a graph is planar, such as the algorithm due to Hopcraft et al. [25] or Boyer et a.l [7]. In Figure 3.4 two formulas are given, one of which no planar version of $G_{\mathbf{X},\mathbf{C}}$ exists, and one of which there does exist such a planar graph.

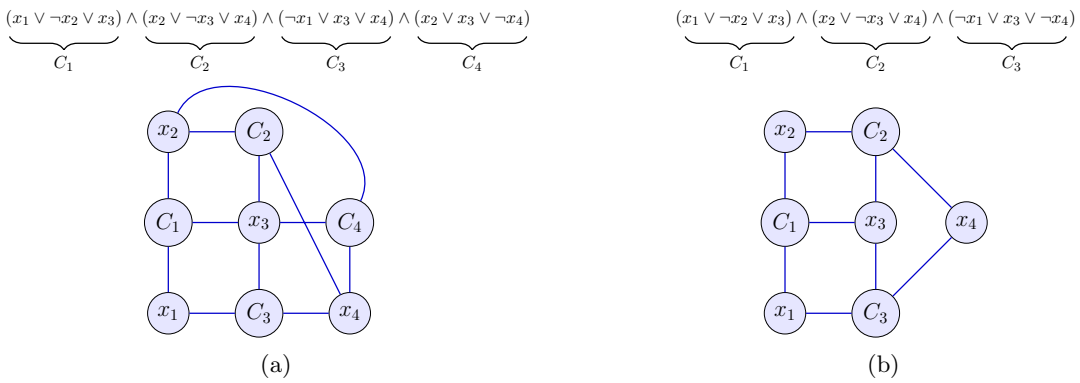


Figure 3.4: Two instances of 3-SAT. The instance in (a) is not planar and the instance in (b) is.

Dahlhaus et al. use a restricted version of P3S in their proof. In this version clauses of size two and three are allowed. Furthermore, they require that for each variable $\mathbf{x}_i \in \mathbf{X}$, the associated vertex in $G_{\mathbf{X},\mathbf{C}}$ has exactly degree three. Another requirement is that for each variable $\mathbf{x}_i \in \mathbf{X}$, both of its literals occurs, each in a different clause. Of course Dahlhaus et al. prove this restricted version of P3S, which we will denote P3Sr, to be an NP-Complete problem as well.

The last problem that we will define here, is the MTC problem. This problem looks like the MULTICUT problem. In the MULTICUT problem we had a graph $G = (V, E)$ and a set of terminal pairs $\mathbf{T}$. Finding a valid solution for the MULTICUT requires finding a minimal set of edges $\mathbf{M}$ that separates each pair of vertices s_i and t_i, where $(s_i, t_i) \in \mathbf{T}$. In MTC, instead of a set of terminal pairs, we have a set of vertices $\mathbf{T} \subseteq V$. A valid solution for MTC is a set of edges $\mathbf{M}$ that separates each terminal $t_i \in \mathbf{T}$ from all other terminals $t_j \in \mathbf{T} \backslash \{t_i\}$. The MTC problem remains NP-Hard on a planar graph with either weighted or unweighted edges, as long as $k = |S|$ is not given. The proof as formulated by Dahlhaus et al. [14] will be given in the next section.

The decision version of MTC is almost the same as the optimization version described above. In this version, besides $G = (V, E)$ and $\mathbf{T}$, a positive integer weight $w(e)$ is associated with each edge $e \in E$. Furthermore, a bound B is given. Instead of searching for the minimal set of edges $\mathbf{M}$ that separates all sinks, it is only required that a set $\mathbf{M}$ exists that separates all terminals in $\mathbf{T}$ and that $\sum_{e \in \mathbf{M}} w(e) \leq B$.

3.3.2 Sketch of proof that MTC is NP-Hard

In this section parts of the proof as given by Dahlhaus et al. [14] that the planar version of MTC is NP-Hard will be repeated. These parts are important for the understanding of Section 3.3.3. Dahlhaus et al. prove that the decision version of MTC is NP-Complete. To prove this, a component design approach is used. For each $\mathbf{x}_i \in \mathbf{X}$, one of the components shown in Figure 3.5 is used to replace the corresponding $\mathbf{x}_i$ in the graph $G_{\mathbf{X},\mathbf{C}}$.

Whenever two x_i literals and one $\neg x_i$ literal of variable $\mathbf{x}_i$ occur in clauses, component (a) will be used, otherwise component (b) is used. The numbers located at the edges are the corresponding weights of the edges. The vertices labelled x_i and $\neg x_i$ will be terminals used in the MTC problem. In (a) the link-vertices $l_{i,1}$ and $l_{i,2}$ will be connected to the clauses that contain the literal x_i and $l_{i,3}$ to the clause that contains the literal $\neg x_i$. For (b) the link-vertex $l_{i,1}$ will be connected to the clause that contains the literal x_i and $l_{i,2}$ and $l_{i,3}$ to the clauses that contain the literal $\neg x_i$.

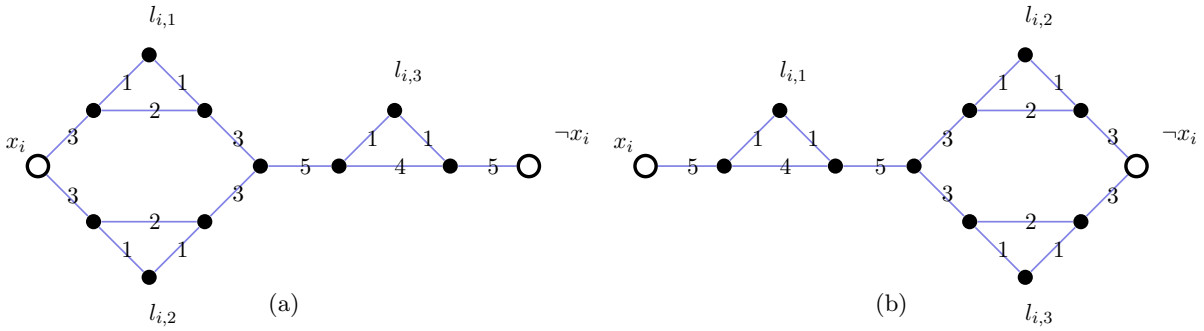


Figure 3.5: The widgets used by Dahlhaus et al. [14]. The widget in (a) is used for $\mathbf{x}_i$ with two x_i literals and one $\neg x_i$ literal. The widget in (b) is used for $\mathbf{x}_i$ with one x_i literal and two $\neg x_i$ literals.

Just like every $\mathbf{x}_i$ was replaced by a component, every $c_j \in \mathbf{C}$ will be replaced by one of the components given in Figure 3.6. Once again the numbers located at the edges are the corresponding weights of the edges and the vertices labelled c_j^+ and c_j^- will be terminals used in the MTC problem. For every clause existing out of 3 literals, the component of (a) will be used. For clauses of size 2 the component of (b) will be used. The link-vertices $m_{j,1}$ through $m_{j,3}$ represent the literals $c_{j,1}$ through $c_{j,3}$ in clause c_j. The link-vertices will be connected to the link-vertices of the components representing the corresponding literal for each variable.

As an example, consider the clause $c_j = x_1 \vee \neg x_2 \vee \neg x_3$, where ther variables $\mathbf{x}_1$ and $\mathbf{x_2}$ have two x_i literals and $\mathbf{x}_3$ only one. Since c_j is a clause existing out of three literals, it will be replaced by the component given in Figure 3.6a. The vertices representing $\mathbf{x}_1$ and $\mathbf{x}_2$ will be replaced by the component as shown in Figure 3.5a (since they have two x_i literals) and the vertex representing $\mathbf{x}_3$ will be replaced by the component as shown in Figure 3.5b. Whenever the literal x_1 is represented by $m_{j,1}$, $\neg x_2$ by $m_{j,3}$ and $\neg x_3$ by $m_{j,2}$, link-edges of weight two have to be added connecting $m_{j,1}$ to either $l_{1,1}$ or $l_{1,2}$, $m_{j,3}$ to $l_{2,1}$ and $m_{j,2}$ to either $l_{3,2}$ or $l_{3,3}$. Furthermore, only one link-edge may be connected to a $m_{j,x}$ or $l_{i,x}$ link-vertex.

The components belonging to clauses and the components belonging to variables are linked together by weight two link-edges. Since the original graph $G_{\mathbf{X},\mathbf{C}}$ was planar and every variable $\mathbf{x}_i$ occurs in at most three clauses, it is possible to link the different components together while keeping the transformed graph planar.

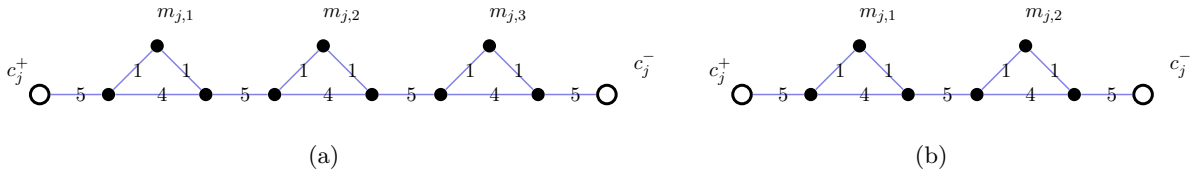


Figure 3.6: The widgets used by Dahlhaus et al. [14]. The widget in (a) is used for clauses with three variables and (b) is used for clauses with two variables.

Dahlhaus et al. continue by proving that when the decision version of the `MTC` problem is solved for the transformed graph with $B \leq 10|\mathbf{X}| + 4|\mathbf{C}|$, a valid solution for `P3Sr` can be obtained from $\mathbf{M}$. While proving this, they also prove a lemma which is important for Section 3.3.3. This lemma concerns all the link structures in the transformed $G_{\mathbf{X},\mathbf{C}}$. A link structure consists out of the vertex-induced subgraph by $m_{j,x}$, the $l_{i,x}$ that $m_{j,x}$ is linked to, the neighbours of $m_{j,x}$ and the neighours of $l_{i,x}$. Such a link-structure is shown in Figure 3.7. Edge (e, f) can have weight two or four, depending on to what link-vertex it belongs to. In this figure $m_{j,x}$ and $l_{i,x}$ are the link-vertices and the edge $(m_{j,x} \rightleftharpoons l_{i,x})$ is the link edge. Dahlhaus et al. prove, when a valid solution for `MTC` is found, that $\mathbf{M}$ contains exactly one of the sets $\{(a, m_{j,x}), (b, m_{j,x})\}$, $\{(l_{i,x}, e), (l_{i,x}, f)\}$ or $\{(m_{j,x}, l_{i,x})\}$. This means that all pairs x_i and $\neg x_i$ are separated from all other terminals and that all pairs c_j^- and c_j+ are separated from all other terminals.

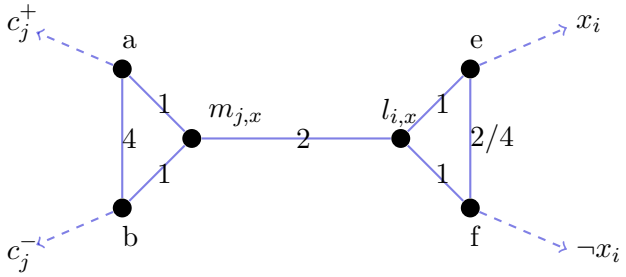


Figure 3.7: A link structure. The edge between $m_{j,x}$ and $l_{i,x}$ is the link-edge, connecting clause structures to variable structures. The edge between e and f has a weight of 2 or 4, depending on what link vertex of a variable it belongs to.

3.3.3 Proof that `MIN-T-MLE` is NP-Hard

To prove that `P3Sr` can be reduced to `MIN-T-MLE`, it is important that the transformed graph obtained from Section 3.3.1 can be reconstructed in a weakly-walkable realistic polygonal environment $\mathbf{P}$. Furthermore, it has to be the case that from $\mathbf{P}$ a $\mathbf{P}$-Graph can be constructed that will yield the same results for `P3Sr` as would be obtained from the transformed graph $G_{\mathbf{X},\mathbf{C}}$. Possible polygonal representations of the variable component and the three variable clause are shown in Figure 3.8.

In these structures, a vertex v with weighted edge degree (which is $\sum_{(v,x)\in E} w(e)$) of two is represented by convex tetragons, of degree five by a convex pentagon, with degree six by a convex hexagon, of degree ten by a convex decagon and of degree eleven by a convex

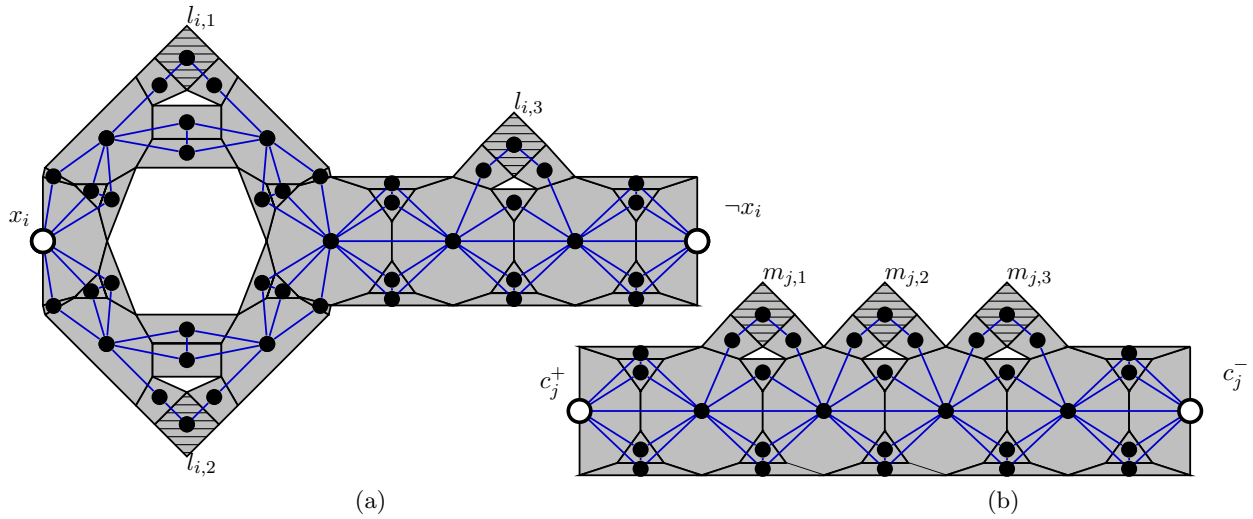


Figure 3.8: Polygonal equivalents of the variable structure (a) and a clause structure (b). The blue lines are the edges belonging to the **P**-Graph and the circles the vertices of the **P**-Graph. The vertices of the **P**-Graph that correspond to the terminals in the components of Dahlhaus et al. are the circles with the labels x_i, $\neg x_i$, c_j^+ and c_j^-.

hendecagon. Triangles and tetragons are used to connect the different polygons to each other. The resulting **P**-Graph has the same properties as the components given by Dahlhaus et al. For example, take a look at Figure 3.9. In this figure, an example is given of how an edge of weight five can be represented. Vertices v_1 and v_2 are vertices that exist in the original components of Dahlhaus, and they are connected by an edge of weight five. The **P**-Graph representation has the same properties as a single weighted edge. Instead of a single edge, five vertex disjoint paths are created connecting v_1 to v_2. As a result, all five paths should be cut if v_1 were to be separated from v_2. The same goes for the entire polygonal representation.

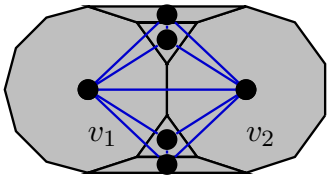


Figure 3.9: A close-up of how a weight five edge can be represented in a **P**-Graph.

The link-edges connecting the link-vertices $l_{i,x}$ to $m_{j,x}$ can be represented using a series of tetragons. Since the link-edges are of weight two, two of such parallel series are needed, attached to the only two free sides of the tetragons $l_{i,x}$ and $m_{j,x}$.

Next, it is necessary to create all the terminals in the **P**-Graph. Whereas in `MTC` it is allowed to pick any set of vertices to be terminals, in a **P**-Graph this is only allowed for vertices belonging to polygons that overlap. Furthermore, in `MTC` all terminals have to be separated, whereas in a **P**-Graph only pairs of vertices have to be separated if that specific pair of vertices belong to overlapping polygons. Therefore, it is necessary to somehow position all $2(|\mathbf{X}|+|\mathbf{C}|)$ terminals such that they all overlap. This can be done by taking $2(|\mathbf{X}|+|\mathbf{C}|)$ polygons centred around the same xz-coordinate. By spacing the polygons using a sufficiently big y difference,

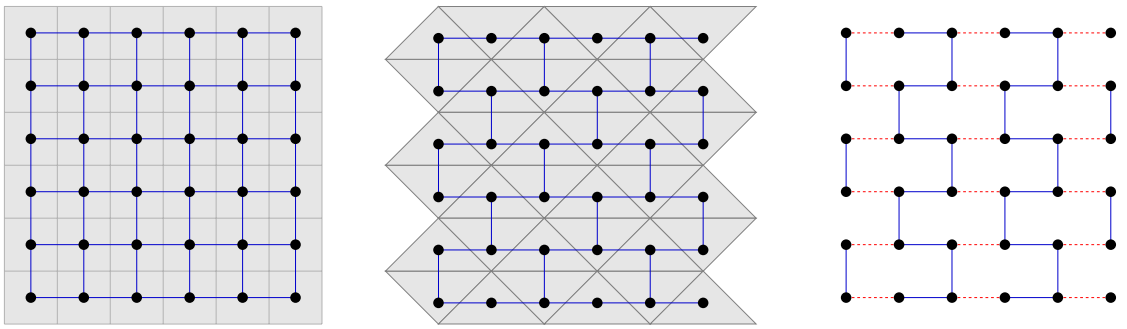
we now have the desired number of terminals that all overlap each other. As a consequence, all these polygons need to be placed in different components when solving MIN-T-MLE. Attaching these polygons, and thus there corresponding terminals, to the terminals x_i, $\neg x_i^-$, c_j^+ or c_j^- can be done by creating a path from each such polyon to the corresponding polygon of x_i, $\neg x_i^-$, c_j^+ or c_j^-. This needs to be done for all x_i, $\neg x_i^-$, c_j^+ and c_j^-. Since the tower exists out of $2(|\mathbf{X}|+|\mathbf{C}|)$ a sufficient number of such polygons exist. To ensure that these paths will not be cut, a sufficient number of parallel paths connecting such a polygon to x_i, $\neg x_i^-$, c_j^+ or c_j^- need to be created to create the effect of a weighted edge. The weight of this edge should be at least ten to ensure these paths will not be cut when solving MIN-T-MLE.

Another thing that needs to be ensured is that the path connecting an overlapping polygon P_o to a terminal is not self-overlapping. The last part that has to be guaranteed is that no path from P_o overlaps any polygon in the clause structure or variable structure it will be connected to, nor one of the three link-paths linking its corresponding clause- or variable-structure to the corresponding structures. The last property is sufficient, since it is already guaranteed using just x_i, $\neg x_i^-$, c_j^+ and c_j^- as terminals that every variable- or clause-structure will be separated from the other structures. Since the newly created overlaps will not change this or change the optimum this will yield the same result as Dahlhaus et al. had for solving P3Sr using planar MTC.

That the reduction given above can be done in polynomial time is not easy to see. It is obvious that every $v \in \mathbf{X} \cup \mathbf{C}$ can be replaced in polynomial time by the corresponding polygonal component. When $n = |\mathbf{X}|+|\mathbf{C}|$, the most complex link-path connecting a variable component to a clause component needs at most $O(2 \times 24n)$ polygons, since that is the most complex shape the components can form. Other $2n$ polygons are needed to create the 'tower' of overlapping polygons, and at most $2n \times O(2 \times 22n)$ polygons are needed to create the paths connecting the 'tower' to their corresponding terminals. The number of overlaps created will be exactly $n^2 + n$ for creating the 'tower' of overlapping polygons and the $2n$ paths leaving the tower will have at most $O(n-1)$ overlaps for each component and at most $O(n)$ overlaps for each link-path. Whenever a large number of paths leaving the 'tower' overlap, at most another $O(((2n)^2+2n) \times 88n^2)$ overlaps will be created. This in turn means that it is possible to construct the required polygonal environment in polynomial time. Therefore, the decision version of MIN-T-MLE is NP-Complete.

3.4 Bounds on the treewidth for P-graphs

From the previous section we know that MIN-T-MLE can be solved by using MULTICUT. Furthermore, we know, if the treewidth ω is bounded for **P**-Graphs, that it is possible to solve MULTICUT, and therefore MIN-T-MLE, using a FPT-time algorithm. The treewidth of a graph G is the width of the minimal width tree decomposition of G. A tree decomposition D of G is a tree with associated to each vertex of the tree a set of vertices from G. Furthermore, when an edge (v, w) exists in G, there should be at least one set in D containing both v and w. Furthermore, for every vertex v contained in G, it should be the case that vertices in D with associated sets containing v should form a connected subtree of D. The width of a tree decomposition D is defined to be the size of the biggest set minus one. Finding a minimal width tree decomposition of a graph is an NP-Hard problem. However, for certain types of graphs, it is possible to determine beforehand what the treewidth of that graph must be. For more information about treewidth, we refer the reader to 'A Tourist Guide through



(a) (b) (c)

Figure 3.10: (a) A $(6, 6)$-grid minor shown on top of an environment $\mathbf{P}$ containing only tetragons. (b) A $(6, 6)$-'brick wall'[3] can be obtained from an environment containing triangles. When the red edges from (c) are contracted, a $(3, 6)$-grid is formed.

Treewidth' by Bodlaender [6].

To determine if a given $\mathbf{P}$-Graph has bounded treewidth, we take a look at graph minors. According to Robertson and Seymour [36], a graph H is a minor of graph G whenever graph G can be reduced to H by:

- Removing vertices that have no edges from G;
- Removing edges from G;
- Contracting edges of G.

One of such minors is the grid minor as depicted in Figure 3.10a. It is the case that whenever a graph contains a (x, x)-grid minor, the treewidth of this graph is at least x [38]. It is easy to see that any $\mathbf{P}$-Graph constructed from an environment $\mathbf{P}$ that may contain tetragons can contain a (x, x)-grid minor. Therefore, for these kinds of environments the treewidth of the associated $\mathbf{P}$-Graph is unbounded.

For an environment $\mathbf{P}$ that may contain triangles, an environment and the corresponding $\mathbf{P}$-Graph are given in Figure 3.10b. This $\mathbf{P}$-Graph once again contains a grid minor. This is demonstrated in Figure 3.10c. When the red dashed edges are contracted, a $(3, 6)$-grid minor is found, which once again contains a $(3, 3)$-grid minor. Therefore, the treewidth of this $\mathbf{P}$-Graph is at least 3. When the environment of Figure 3.10b is increased in size, so will the treewidth. Therefore, for this type of environment the treewidth is unbounded.

When considering all other possible types of environments[2], the treewidth remains unbounded. According to Robertson and Seymour [37], every planar graph has a (x, x)-grid minor. Since every $\mathbf{P}$-Graph can have a planar graph minor, this means that any type of $\mathbf{P}$-Graph has an unbounded treewidth. As a consequence, it is not possible to use FPT-time algorithms (e.g. the algorithms described by Guo [23]) for the `MULTICUT` problem to solve the `MIN-T-MLE` problem in polynomial time.

[2]That is, environments containing only pentagons or pentagons and tetragons or pentagons, tetragons and triangles, or only hexagons, etc.

Chapter 4

Finding subproblems and reductions

Since finding a MIN-T-MLE is hard from a computational point of view, it is important to take a look at possible subproblems and reductions of the original problem instance. In Section 4.1, we will explore a group of subgraphs of a **P**-Graph. For this family of subgraphs, solving the MIN-T-MLE problem for subgraph G' of G will result in a decomposition of G' into a multi-layered environment (MLE) $\mathbf{L}_{G'}$. For this decomposition, each layer in $\mathbf{L}_{G'}$ will form a connected component in an optimal MLE for G. Another group of subgraphs will be discussed in Section 4.2. This group of subgraphs does not necessarily preserve the optimal MLE, but is less computational intensive.

Section 4.3 will discuss methods that can be used to reduce the number of vertices, edges and overlaps in a **P**-Graph. Two of these methods were introduced by Saaltink [40], which only enabled edge and vertex reductions. The new methods introduced in this section also enable the removal of overlaps.

4.1 Stable subgraph

In this section we try to find a family of subgraphs of G which we can use to find an MLE for G that is optimal with respect to the number of transfers. More precisely, we are searching for subgraphs G' of G for which, when a minimal MLE $\mathbf{L}$ has been found for G', each individual layer in $\mathbf{L}$ will form a connected component in a minimal MLE for G. In this section, an optimal MLE is always with respect to the number of transfers.

4.1.1 Definition

A subgraph G' of G for which such an MLE exists, will be referred to as a *stable subgraph*. Furthermore, we assume that this graph is vertex-induced. This means that the vertex set V' is a subset of V. Moreover, if a pair $v, w \in V'$ forms the endpoints of an edge $(v \rightleftharpoons w) \in E$, this edge also exists in the subgraph $G(V')$. The same goes for the set of overlaps O' of $G(V')$. Whenever two vertices $v, w \in V'$ correspond to an overlap $(v \Updownarrow w) \in O$, $(v \Updownarrow w)$ will also exist in $G(V')$. The corresponding polygonal environment will be denoted $\mathbf{P}(V')$. Whenever there is only one subgraph in play, the notation G' will be used instead of $G(V')$ and $\mathbf{P}'$ instead of $\mathbf{P}(V')$.

Definition 8. *Given a* $\mathbf{P}$*-graph* $G = (V, E, O)$ *and a vertex-induced subgraph* G' *of* G*. When there exists an optimal MLE* $\mathbf{L}_{G'}$ *of* G' *and an optimal MLE* $\mathbf{L}_G$ *for* G *for which each layer* $L_i \in \mathbf{L}_{G'}$ *forms a connected component in a layer* $L_j \in \mathbf{L}_G$*,* G' *is a* **stable subgraph**.

4.1.2 Application

When such a subgraph G' has been found, as well as the desired MLE $\mathbf{L}_{G'}$ for which each $L \in \mathbf{L}_{G'}$ is a connected component in a layer $L \in L_G$, it is quite simple to use $\mathbf{L}_{G'}$ for finding an optimal MLE for G. This can be done in the following three steps. First, for each $L_i \in \mathbf{L}_{G'}$ the vertices in L_i are replaced by one single vertex v_i.

During the second step, for each layer L_i the edges E_v for each $v \in L_i$ are inspected. If there is an edge $(v \rightleftharpoons w)$ where $w \in L_j$ and $j \neq i$, an edge $(v_i \rightleftharpoons v_j)$ is created. When there exists an edge $(v \rightleftharpoons w)$ where $w \in L_i$, the edge is discarded. The last type of edge is an edge neither in L_i or L_j where $j \neq i$. When such an edge $(v \rightleftharpoons w)$ is found, it is replaced by an edge $(v_i \rightleftharpoons w)$.

In the third and final step, for each layer L_i the overlaps O_v for each $v \in L_i$ are inspected. Since L_i is a layer in a valid MLE, we already know that no overlap $(v \Updownarrow w) \in O_v$ exists where $w \in L_i$. If such an overlap would exist, the found MLE would be invalid. Whenever an overlap $(v \Updownarrow w) \in O_v$ with $w \in L_i$, $j \neq i$, it is replaced by $(v_i \Updownarrow v_j)$. In the last situation where $(v \Updownarrow w) \in O_v$, but $w \neq V'$, the overlap is replaced by $(v_i \Updownarrow w)$.

In the resulting graph $G_R = (V_R, E_R, O_R)$, the number of edges can be further reduced when there exist multiple $(v \rightleftharpoons w) \in E_R$. When we consider finding an optimal MLE as finding a minimal cut set $\mathbf{M}$ for the `MULTICUT`-problem as described in Section 3.2, the overlaps become terminal pairs and the edges in $\mathbf{M}$ will form transfers. From `MULTICUT`, we know that either no edges between v and w are cut, or all edges between v and w are cut. Whenever some edges between v and w are cut, $|\mathbf{M}|$ could be reduced by removing all edges $(v \rightleftharpoons w)$ from $\mathbf{M}$, since these edges do not separate any terminal pairs. Therefore, all n edges between v and w can be replaced by a single edge $(v \rightleftharpoons w)$ of capacity n.

The size of O_R can be further reduced as well, when switching to the `MULTICUT` framework. When there exist multiple overlaps $(v \Updownarrow w) \in O_R$, they can be replaced by a single overlap $(v \Updownarrow w)$. This would still force the terminals v and w to be separated and will not change the value of the solution when there would exist multiple of these (v, w) terminal pairs in `MULTICUT`.

Because of the assumption that an MLE $\mathbf{L}_{G'}$ was found for which each layer would form a connected component an $L \in \mathbf{L}_G$, it is easy to see that this will not influence the optimal solution $\mathbf{L}_G$ for G. Merging the vertices as described before only ensures that these vertices will end up in the same layer. Because of the way edges and overlaps are removed and replaced, the same paths connecting terminals in `MULTICUT` still exist, as well as the same terminal pairs.

4.1.3 Properties of a stable subgraph

In the previous section we have seen how a stable subgraph of G can be used to search for an optimal MLE for G itself, however two questions remained unanswered. Firstly, it is nice to know which of the solutions for the `MIN-T-MLE`-problem solved for G' should be used. One possible way would be to find all minimal MLEs for each subgraph and use each of these MLEs

as described above. After these solutions have been used to find a MLE for G, the solutions yielding the minimal number of cuts is used. However, this might result in a very large number of computations. If a minimal MLE for G' has $|\mathbf{T}| = t$, up to $\binom{|E'|}{t}$ different solutions can exist. Therefore, in this section we will only take a look at subgraphs of G for which all optimal MLEs can be used for finding the optimal solutions for G. These subgraphs will be referred to as *super stable subgraphs*. Secondly, we need to know under what circumstances all minimal MLEs of subgraph G' can be used for finding the minimal MLE of G. This will be discussed next.

In this section one property will be given. When this property holds for a subgraph it is guaranteed to be super stable, however there may exist subgraphs that do not satisfy this property that are still super stable. In this section, the notation for a simple path p from v to w will be $[v] \rightarrow [w]$. A simple path starting with a fixed sequence of vertices $[v, w]$ and ending in another fixed sequence of vertices $[x, y]$ will be denoted as $p = [v, w] \rightarrow [x, y]$. The first vertex of such a path is $p_0 = p_v = v$. The second vertex of such a path is $p_1 = p_{v+1} = p_w = w$ and the last element is $p_y = y$. The property concerns edges $(a \rightleftharpoons b) \in E'$ and overlapping vertices v and w.

Property 1. *For G' to be super stable, all vertices on all paths $p = [v] \rightarrow [a, b] \rightarrow [w]$ with $(v \Updownarrow w) \in O$, $(a \rightleftharpoons b) \in E'$ should be in V'.*

Lets first consider the edges in a super stable subgraph.

Lemma 1. *Assume we have an edge $e = (x \rightleftharpoons y) \in E$ for which no path $p = [v] \rightarrow [x, y] \rightarrow [w]$ in G with overlapping v and w exists. When e is removed from or added to a super stable subgraph G', G' remains super stable.*

Proof. Since no $p = [v] \rightarrow [x, y] \rightarrow [w]$ in G exists for overlapping v and w, edge e will never separate any overlapping pair $(v \Updownarrow w) \in O$ in G or in G'. This means that e will never be part of the cut set $\mathbf{M}$ in a minimal solution. Therefore, when e is added to or removed from G', the minimal cut set for G' will not change. □

This means that all these edges can be removed from a subgraph of G', while still remaining super stable. This also means that only edges that have paths connecting overlapping vertices have to be in a super stable subgraph. If such an edge is in a super stable subgraph, it is necessary that all simple paths connecting its overlapping vertices are in the stable subgraph. An example of that is given in Figure 4.1. In this figure, $\{c, d\}$ could form a subgraph when property 1 would not be necessary. However, this subgraph does not contain any conflicting vertices and will therefore be in one single layer in the MLE for $G(\{c, d\})$. However cutting the edge $(c \rightleftharpoons d)$ is the optimal solution for G, separating both $(a \Updownarrow b)$ and $(e \Updownarrow f)$.

Lemma 2. *When property 1 holds for a subgraph G', we have enough information to decide if an edge $e = (x \rightleftharpoons y) \in E'$ has to be cut.*

Proof. If property 1 holds, all simple paths connecting overlapping vertices that pass through e are also in G'. This also means that all possible ways of separating these overlapping vertices are in G'. Consider an overlapping pair of vertices $(v \Updownarrow w) \in O'$ that is connected in G' through e. This would also mean that all possible $[v] \rightarrow [x, y] \rightarrow [w]$ paths are in G'. If on these paths no edge $e' = (x' \rightleftharpoons y')$ exists that has a $[v'] \rightarrow [x', y'] \rightarrow [w']$ with $(v' \Updownarrow w')$, $v' \notin \{v, w\}$ and $w' \notin \{v, w\}$, it is easy to see that the decision can be made for e if it has to be

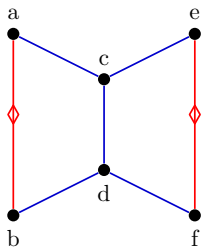


Figure 4.1: A situation that shows why property 1 is necessary.

cut or not. The addition of any of these edges to the cut set can only separate v from w, not any other pair of overlapping vertices. Furthermore, these paths have to be cut to separate v from w, meaning that they have to be cut and that the cut can not be influenced by anything not currently in G'.

If on these paths such an edge e' does exist, it would mean that the corresponding v' and w' would also be part of G', as well as all $[v'] \to [e'] \to [w']$ paths. If on these paths no edge e'' exists that have a $[v''] \to [e''] \to [w'']$ with $(v'' \Updownarrow w'')$, $v'' \notin \{v, w, v', w'\}$ and $w'' \notin \{v, w, v', w'\}$, all information for deciding if e' should be cut is known, and, therefore, all information to decide if e should be cut. This process can be repeated to show that property 1 is sufficient to decide whether e should be cut in both G' and G. □

The downside of restricting ourselves to subgraphs that adhere to property 1 is that the subgraphs tend to be very big. For this reason we will also take a look at a different type of subgraph.

Furthermore, checking if an edge $(v \rightleftharpoons w)$ should be in G' takes quite some time. To determine if an edge $e = (v \rightleftharpoons w)$ is on a path connecting two overlapping vertices $(x \Updownarrow y) \in O'$, we can utilize a max-flow computation [3]. In this computation, all the edges of the original graph will have capacity 1. A source s is added with edges $(s \rightleftharpoons v)$ and $(s \rightleftharpoons w)$, both with capacity 1. The sink vertex t will have the edges $(t \rightleftharpoons x)$ and $(t \rightleftharpoons y)$ also both with capacity 1 where $(x \Updownarrow y) \in O'$. If the maximal flow is two, e is on a simple path connecting x to y. Since it can be the case that e does not connect all $|O'|$ overlapping pairs of vertices in G', at most $|O'|$ of these tests have to made for a single edge.

4.2 Topo-Forest

In this section, another type of subgraph that looks very promising for certain types of environments is discussed. This subgraph relies on the idea that most **P**-Graphs will be constructed from environments that look like conventional buildings. The usefulness of this subgraph for solving `MIN-T-MLE` will be tested in Chapter 7 and beyond.

4.2.1 Definition

A Topo-Forest (TF for short) is a subgraph $G' = (V', E', O')$ of G induced by $V' \subseteq V_O$. Here V_O is the set of vertices that have an overlap with at least one other vertex. A pair of vertices $v \in V_O$ and $w \in V_O$ is in the same TF if and only if there exists a path $p = [v] \to [w]$ that

only visits vertices in V_O. Furthermore, for each pair of succeeding vertices p_i and p_{i+1} on path p, there has to exist an edge $(p_i \rightleftharpoons p_{i+1}) \in E$ and/or an overlap $(p_i \Updownarrow p_{i+1}) \in O$.

Definition 9. *A Topo-Forest is a subgraph $G' = (V', E', O')$ of G induced by V' where:*

1) $V' \subseteq V_O$*;*

2) *v and w are in the same TF if and only if there exists a path $p = [v] \rightarrow [w]$ of length n such that:*

- $p_i \in V_O \ \forall 1 \leq i \leq n$*;*
- *p_i and p_{i+1} are connected or overlap.*

Because of the definition of a TF, either v and all members of O_v are in V', or neither v and O_v are in V'. An example showing two TFs is given in Figure 4.2.

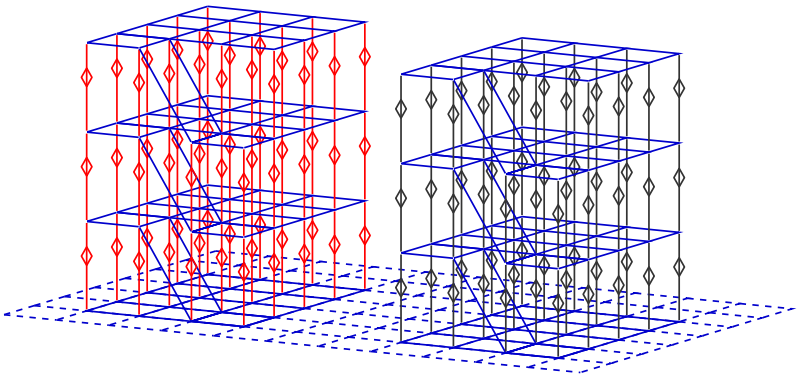


Figure 4.2: Two TFs in one **P**-Graph. The red overlaps and corresponding vertices form one TF, the black overlaps and their corresponding vertices the second TF. The dashed edges are the edges that are not part of any TF.

As can be readily seen, this type of subgraph is not a stable subgraph. It is possible that a transfer for the optimal MLE is within a TF while the corresponding edge is not cut when solving the `MIN-T-MLE` for this particular TF. An example of this is given in Figure 4.3. In this example, using the decomposition into a minimal MLE for a subgraph as described in Section 4.1.2 will yield an MLE with $|\mathbf{T}| = 9$, while the minimal MLE for G has $|\mathbf{T}| = 7$.

4.2.2 Finding a TF

Finding all TFs in a graph G is straightforward. First a graph $G' = (V', E')$ is constructed, where V' contains all the vertices in V_O. When v and w are vertices in V', an edge $(v \rightleftharpoons w)$ is added to E' for all $(v \rightleftharpoons w) \in E$ and all $(v \Updownarrow w) \in O$.

The vertices of each connected component in G' are the vertices belonging to each TF in G. The connected components of G' can be found by running c times a BFS or DFS searche, where c is the number of connected components in G'. A description of such an algorithm is given in Algorithm 1 and has complexity $O(|V'| + |E'|)$.

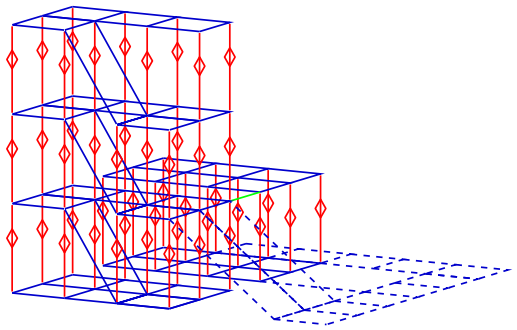


Figure 4.3: An example TF. The dashed edges are part of the remainder of the graph. The decomposition of the TF into $\mathbf{L}_{G'}$ consist of all the individual floors (5 layers). The optimal decomposition of G, however, is found by cutting the ramps in the TF and the green edge, which resutls in 7 transfers. Using the decomposition $\mathbf{L}_{G'}$ would result in all ramps to be cut and three transfers outside of G', resulting in a total of 9 transfers.

Algorithm 1 Pseudocode for finding connected components. Pseudocode for BFS(G, v) is given in Algorithm 2.

Input: $G = (V, E)$: A graph
1: $C \leftarrow$ new List
2: **for** $v \in V$ **do**
3: **if** v has been previously visited by BFS **then**
4: **continue**
5: **end if**
6: $C.insert$(BFS(G, v))
7: **end for**
8: **return** C

4.2.3 Application

Since an optimal MLE for a given TF t might not be suitable for use as described in Section 4.1.2, we need to check each `MIN-T-MLE` decomposition of a TF t to see if it can be used when searching for the optimal solution for G. How this can be done is described in this section.

Suppose that the vertices of a graph G are split into the set of all TFs (denoted **TF**) and a set containing the remainder of the graph (denoted **GP**). In **TF**, multiple TFs are stored. Because of how a TF is defined, we know that the vertices contained in a TF t are a subset of V_O. Furthermore, we know that the following lemma holds:

Lemma 3. *A vertex $v \in V$ is assigned to at most one TF.*

Proof. Suppose we have vertex v. Vertex v is assigned to a TF t and t'. First of, we know that every TF is connected through its edges and overlaps because of its definition. This means that any vertex $u \in t$ and $w \in t'$ is connected to v. Therefore, u is also connected to w on a path through $E \cup O$ through vertices only in V_O in G. Therefore, it must be the case that u and w are in the same TF and therefore $t = t'$. □

Algorithm 2 Pseudocode for a BFS implementation

Input: $G = (V, E)$: A graph;
v: Starting point of search for connected component.

```
1: C ← new List
2: Q ← new Queue
3: Mark v as visisted
4: C.insert(v)
5: Q.insert(v)
6: while ¬Q.empty() do
7:     v' ← Q.dequeue()
8:     for all neighbours n of vertex v' do
9:         if n not yet visited then
10:            Mark n as visited
11:            C.insert(n)
12:            Q.insert(n)
13:        end if
14:    end for
15: end while
16: return C
```

We assume that for each individual TF, the `MIN-T-MLE` problem has been solved, which means each TF t is decomposed in a set of layers $\mathbf{L}_t$. A layer $L_{i,t} \in \mathbf{L}_t$ overlaps another layer l, if there is at least one vertex in $L_{i,t}$ and one vertex in l that overlap. In the same way, $L_{i,t}$ is connected to l whenever there is at least one vertex in $L_{i,t}$ connected to at least one vertex in l. The vertices from i connecting layer i to layer j are in the set $\mathbf{C}_{i,j}$. The set of vertices contained within a layer i of a TF that have edges connecting this layer to vertices contained within $\mathbf{GP}$, is denoted $\mathbf{F}_{i,t}$.

Now, given a TF t with an optimal decomposition into $\mathbf{L}_t$, it is readily seen that whenever $l \in \mathbf{L}_t$ has no edges going outside of the TF ($|\mathbf{F}_{t,l}| = 0$), l is a stable layer as long as all of the layers bordering l are stable.

Lemma 4. *Given are a TF t, its optimal MLE $\mathbf{L}_t$, and a layer $l \in \mathbf{L}_t$ with $\mathbf{F}_l = \emptyset$. If for all layers $m \in \mathbf{L}_t$ for which there exists a vertex $v \in l$ such that $v \in \mathbf{C}_{l,m}$, m is stable, than l is also stable.*

Proof. The first thing to note is that l has no edge to any vertex in $\mathbf{GP}$. This means no edges between vertices in $\mathbf{GP}$ and l will be created when searching for the `MIN-T-MLE` for G. Therefore, the only influence on the distribution of the vertices of l are from the neighbouring layers, which are the layers $m \in \mathbf{L}_t$ for which $w \in m$ and $w \in \mathbf{C}_{m,l}$.

Since there are neighbouring layers, it must be the case that there is a vertex $o \in m$ and $v \in l$ that overlap, where m is a neighbouring layer. If such a vertex-pair does not exist, this would mean that $\mathbf{L}_t$ is not optimal since the layers m and l can be merged. This holds for all the neighbouring layers of l.

The vertices in each neighbouring stable layer will end up in the same layer of $\mathbf{L}_G$, since that is the definition of stable layers. Since the overlapping vertices from the neighbouring layers were already separated in a minimal way in t, this is also minimal for G. □

From here on, we can prove the following lemma:

Lemma 5. *Given a TF t and its optimal MLE $\mathbf{L}_t$. If all $l \in \mathbf{L}_t$ for which $|\mathbf{F}|_{t,l} > 0$ are stable, all $m \in \mathbf{L}_t$ for which $|\mathbf{F}|_{t,m} = 0$ must be stable too.*

Proof. Since all m are optimally separated from their neighbouring layers, several situations can arise. First, all neighbouring layers of m can have $|\mathbf{F}|_{t,l} > 0$. That situation has already been discussed in Lemma 4.

For all other situations the following holds. If the layers l are stable, it once again means that all vertices from l end up as a connected component in a single layer of the optimal MLE for G, otherwise l was not stable. If a layer l has a neighbouring layer m with $|\mathbf{F}|_{t,m} = 0$, it cannot change the location of the cut between l and m, since this was already optimal. □

Next we need to know under which circumstances a layer l in a TF with $|\mathbf{F}_l| > 0$ is stable. For these layers, the following situations can occur:

1) There exists another layer $m \in t$ with $|\mathbf{F}_m| > 0$, $l \Updownarrow m$ and l and m are connected through $G \backslash t$. On the path(s) connecting l and m pairs of vertices $(v \Updownarrow w)$ exist;

2) There exists another layer $m \in t$ with $|\mathbf{F}_m| > 0$, $l \Updownarrow m$ and l and m are connected through $G \backslash t$. On the path(s) connecting l and m no pair of vertices $(v \Updownarrow w)$ exists;

3) There exists another TF t' with two layers m and n that are connected through path p with at least two consecutive vertices in l.

For a decomposition of a TF to be usable, all of the above properties should not exist or none of the situations should enforce a different decomposition of t. What this means is explained in the next paragraphs.

When the first situation occurs, it does not force the decomposition of t to change. On the path connecting layers l and m, a subpath $[v] \to [w]$ exists with $(v \Updownarrow w)$. Therefore, this subpath is guaranteed to be cut in any MLE for G and the layers l and m do not need to change.

The second situation where no pairs of vertices exist on a path is not that easily resolved. Suppose such a path p is connected to layer l by vertex v and to layer m by vertex w, as is shown in Figure 4.4. This path only visits vertices in $V \backslash V_t$, where V_t are the vertices of topoforest t. When the vertices v and w overlap each other as in (a), there has to exist a cut disconnecting v and w outside of the layers l and m, otherwise v and w would not be separated.

Whenever a path connecting the two layers exists such that the vertices do not overlap, it can be the case that the layers l and m are not stable. An example of this is given in Figure (b). In this figure there exists a path $[v] \to [w']$ where v and w' do not overlap. Therefore, we have no guarantee that the path $[v'] \to [w']$ will be cut outside of t. For that reason we need to check where this path will be cut.

This can be done by using the max-flow min-cut theorem. Firstly, it is necessary to calculate how expansive it would be to cut all $[m] \to [l]$ paths that have no pair of overlapping vertices on them. This can be done by taking all the $\mathbf{F}_m$ vertices as sources and all the $\mathbf{F}_l$ vertices as sinks. The size of this cut, let us call it C, is an upper bound of the cuts needed to separate layer m from l in G. Secondly, we need to check what the minimal cut between

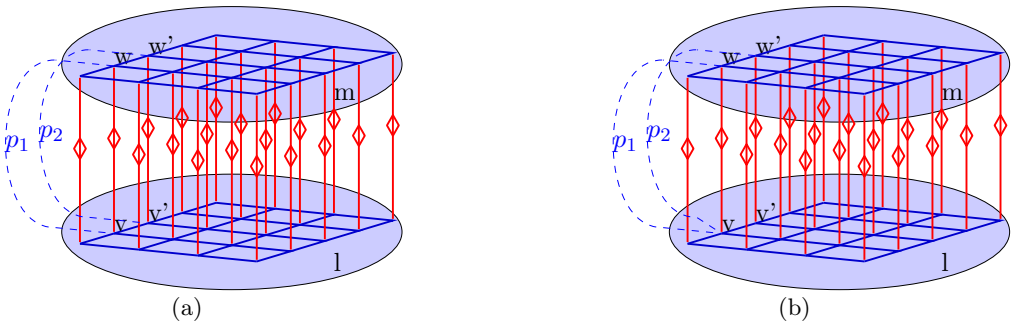


Figure 4.4: The layers l and m with vertices v, v', w and w'. The layers l and m are connected by two paths. The remainder of the graph is not shown. In (a) the two layers are connected by paths $p_1 = [v] \to [w]$ and $p_2 = [v'] \to [w']$. In (b) the two layers are connected by paths $p_1 = [v] \to [w]$ and $p_2 = [v] \to [w']$.

the vertices from l that overlap the vertices in m is. This can once again be done by a simple max-flow min-cut algorithm. If the value found here is smaller than C, this means that l and m are not guaranteed to be stable, since there is a cut separating the overlapping vertices from l and m using edges in l and m. If the cut found is equal to C, the means that the vertices can be cut using the vertices found when searching for the cut separating $\mathbf{F}_m$ from $\mathbf{F}_l$ and therefore will only depend on the other two situations.

Whenever the last situation occurs, vertices in one layer l of a TF t are part of a path connecting two overlapping layers m and n of a different TF t'. In some cases this can force the layer l of t to be split in to two or more components. This happens whenever the 'cheapest' way to separate m and n is by cutting some of the edges in l. Whenever there is only one pair of such layers that have a path connecting them through l, it is relatively easy to check if l is unstable or not. First the minimal cut of m and n is calculated. Whenever this value is smaller than the minimal cut found when the cost of cutting edges contained within l is increased to a high enough value, l is unstable.

Whenever there exist more pairs of layers from possibly multiple TFs that are connected with a path through l, the above method will not work. The reason for this is that this would result in a new instance of multicut. Therefore, whenever this case occurs the TF should be discarded.

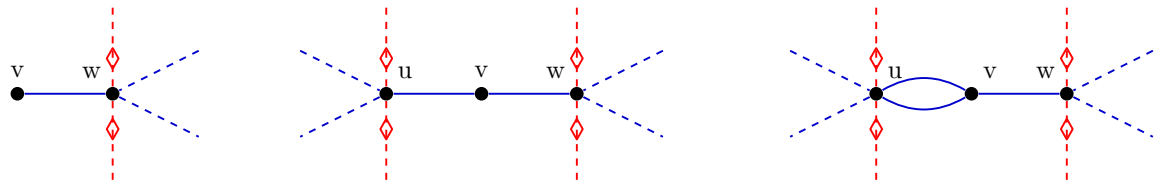
4.3 Reductions

In this section the set of neighbours of v is still $\mathbf{N}_v$ and the set of vertices v has an overlap with is still $\mathbf{O}_v$.

The last thing to note is that until now, it was assumed that the **P**-Graph was an unweighted graph with no capacities assigned to edges. In the upcomming section, this will change. The edges of a **P**-Graph now have a capacity. Before any reduction has been applied to a graph, the capacity of each edge equals one.

4.3.1 Reducing the number of edges

In his master's thesis [40], Saaltink already defines two graph simplifications which we will name `1-CONTRACT` and `2-CONTRACT`. The first reduction is only applicable for a vertex v with $|\mathbf{O}_v| = 0$ and $|\mathbf{N}_v| = 1$. When this is the case this vertex can easily be ignored. There is no reason to add this vertex to any other layer than the layer of its only neighbour. A situation where this operation can be applied is shown in Figure 4.5a.



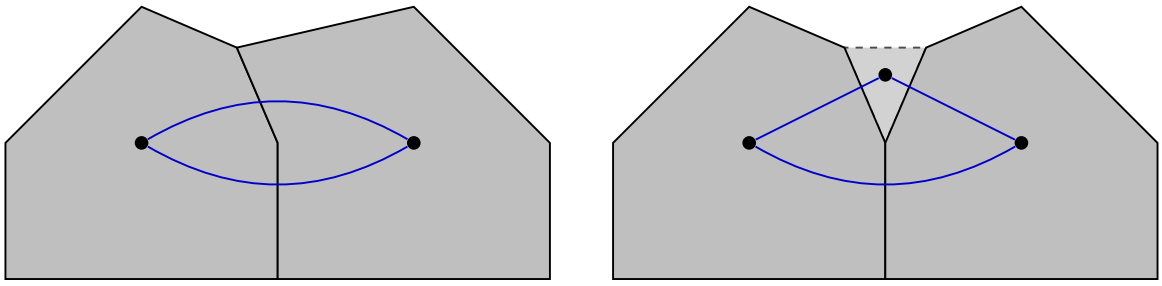
(a) (b) (c)

Figure 4.5: Figure showing situations in which different operations can be applied. The dashed edges and overlaps depict possible overlaps and edges to the remainder of the graph. (a): The vertices v and w can be contracted using `1-CONTRACT` (b): The edges (u, v) and (v, w) can be replaced by a single edges (u, w). Furthermore, the vertex v can be removed. (c): The two edges (u, v) can be merged into a single edge with as capacity the sum of both edges.

The second graph simplification applies to a vertex with $|\mathbf{N}_v| = 2$ and $|\mathbf{O}_v| = 0$. Assume the two neighbours of v are x and y. Since this vertex v has no vertices it overlaps with, contracting v and x or v and y will not induce any new overlaps. The newly created edge (x, y) does not create any new connections between x and y, nor the minimal cost of separating x and y. A situation where this operation can be applied is shown in Figure 4.5b.

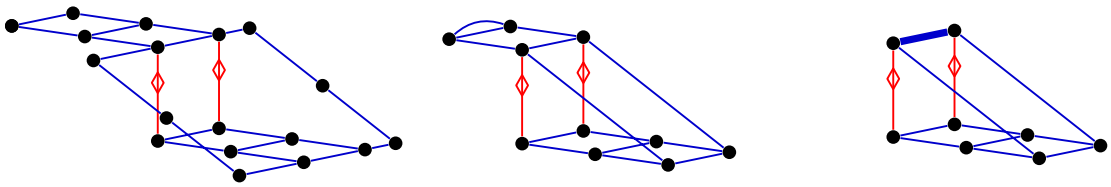
Because of the restrictions of the underlying polygonal environment, some situations cannot exist. Two vertices with two different edges connecting them is one of these situations. This situation is shown in Figure 4.6a. In this figure, the polygon located on the right is concave, while we require that all the polygons are convex. However, when we apply the `2-CONTRACT` operation on an environment as shown in Figure 4.6b, the reduced graph would contain two vertices with two different edges connecting them. Whenever this happens, some edge-reduction operations will not be applied, while in fact they can be applied. An example of a situation like this is given in Figure 4.5c. In this situation it would be nice if we could apply the `2-CONTRACT` operation, but this is not possible since $|\mathbf{N}_v| = 3$, not two. A simple solution solving this problem is merging all the double edges and combining their respective capacities. This method is called `E-REDUCE`. After this operation has been applied to Figure 4.5c, `2-CONTRACT` can be applied.

To enable the `2-CONTRACT` operation to handle edges of different weights, we need to change it a little. This version takes the minimum capacity of either one of the edges. The application of a sequence of the above mentioned operations is illustrated in Figure 4.7.



(a) (b)

Figure 4.6: In (a) the right polygon may not exist in **P** since it is concave. Figure (b) shows a situation where an identical graph would emerge after applying `2-CONTRACT`.



(a) (b) (c)

Figure 4.7: An example of the new contraction operation. In (a) the original **P**-Graph is given, in (b) the contractions possible with the old contraction operator and in (c) the contraction with the new contraction operator. The thick edge is of capacity 2.

4.3.2 Reducing the number of overlaps

A logical next step is considering when overlaps can be removed from the graph. This situation will be subdivided in three categories. First we have the trivial cases, where there is only one possible cut when two vertices have to be separated. Next there is the case of vertices with edge degree 2 that do have overlaps. In this scenario several overlaps can be removed under specific circumstances. The last scenario contains a general approach to detecting if a certain overlap is removable. Although the algorithm belonging to this approach is easy to understand, it is rather computational intensive.

Trivial case

A case is considered trivial when there exists only one simple path $[v] \rightarrow [w]$ and both $(v \rightleftharpoons w) \in E$ and $(v \Updownarrow w) \in O$. When this happens, there is only one possible way to separate v and w, which is done by cutting the edge $(v \rightleftharpoons w)$. Keeping this cut fixed during the entire search process will not influence the optimal decomposition of G into a MLE, since it is the only way to separate v from w. Since this action guarantees that v and w are separated, the overlap $(v \Updownarrow w)$ can be removed from O. When other paths do exist, the cut still remains necessary, however the overlap cannot be removed. The operation of removing the edge between overlapping vertices v and w is called `T-CUT`.

(a) (b)

Figure 4.8: Two trivial situations considering overlaps. In (a) the edge $(v \rightleftharpoons w)$ has to become a transfer, since v and w need to be in separate components. (b) depicts a situation where all paths from v to w are blocked by the single neighbour of v.

Furthermore, when a vertex v has edge degree 1 and $(v \Updownarrow w)$ holds, the overlap can be removed when all the neighbours of w are overlapped by the single neighbour of v. These two situations are shown in Figure 4.8. This operation is known as `1-REMOVE`.

Vertices with edge degree 2

When considering a vertex v with edge degree 2 and its corresponding overlaps, some of these overlaps might be removed. Assuming an overlap $(v \Updownarrow w)$ exists, this overlap can be removed when on all paths connecting v and w, pairs of vertices $(x \Updownarrow y)$ exist. These situations will be treated in Section 4.3.2.

Another situation where an overlap $(v \Updownarrow w)$ can be removed, is given in Figure 4.9a. In this situation all edges have capacity one and both v and w have edge degree two. There are two types of paths connecting v and w. The first type of paths are the paths $p = [v, n_v] \to [n_w, w]$, where $(n_v \Updownarrow n_w) \in O$. For these paths the following lemma holds:

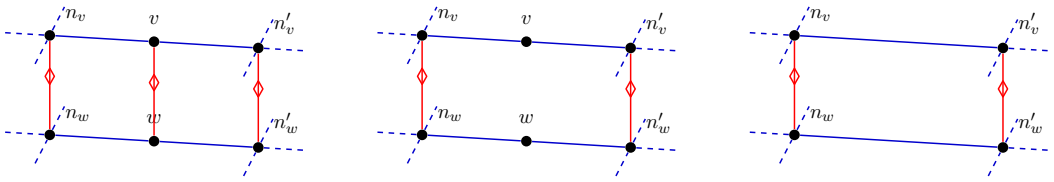
Lemma 6. *Given v, w, $(v \rightleftharpoons n_v)$, $(w \rightleftharpoons n_w)$ and $(n_v \Updownarrow n_w)$. All paths $p = [v, n_v] \to [n_w, w]$ will be cut in a valid `MIN-T-MLE`.*

Proof. On all $p = [v, n_v] \to [n_w, w]$ paths, the subpaths $p' = [n_v] \to [n_w]$ exist. Since $(n_v \Updownarrow n_w)$, on all p' paths at least one edge has to be part of the cut set, separating n_v from n_w and also cutting all $[v, n_v] \to [n_w, w]$ paths. □

The second type of paths are the paths $p = [v, n_v] \to [n'_w, w, n_w]$. On these paths $(n_v \Updownarrow n_w)$ and $(n'_v \Updownarrow n'_w)$. For these paths the following lemma holds:

Lemma 7. *Given v, w, $(v \rightleftharpoons n_v)$, $(w \rightleftharpoons n_w)$, $(w \rightleftharpoons n'_w)$, $(n_v \Updownarrow n_w)$. If a path $p = [v, n_v] \to [n'_w, w, n_w]$ is cut by removing edge $e = (w \rightleftharpoons n_w)$ in an optimal MLE, the value of the MLE will not change when e is replaced by $(w \rightleftharpoons n'_w)$.*

Proof. In an optimal MLE, we know that a path $p = [v, n_v] \to [n'_w, w, n_w]$ is cut, since $(n_v \Updownarrow n_w)$. If this path was not cut, the solution would be invalid. When the optimal cut contains the edge $e = (w \rightleftharpoons n_w)$, it cuts all paths containing the subpath $[n_w, w, n'_w]$. Replacing e by $(w \rightleftharpoons n'_w)$ will still cut all these paths. □



(a) (b) (c)

Figure 4.9: An example of when overlaps can be removed when vertices have edge degree 2.

(a) (b)

Figure 4.10: An example of when overlaps can be removed when vertices have edge degree 2.

Using Lemmas 6 and 7, we can now prove that the overlap $(v \Updownarrow w)$ of Figure 4.9 can be removed.

Proof. By Lemma 6 all paths $[v, n_v] \to [n_w, w]$ and $[v, n'_v] \to [n_w, w']$ will be cut even if v does not overlap w. Furthermore, we know that all paths of type $p = [v, n_v] \to [n'_w, w, n_w]$ will be cut in a valid MLE. If these paths are cut using edges different from an edge in $X = \{(v \rightleftharpoons n_v), (v \rightleftharpoons n'_v), (w \rightleftharpoons n_w), (w \rightleftharpoons n'_w)$, v and w are separated.

However, if such a path is cut using an edge from X, it can be the case that v and w are not separated, for example when the path $p = [v, n_v] \to [n'_w, w, n_w]$ is cut by the edge $(n_w \rightleftharpoons w)$. However, by Lemma 7 we know that this edge can be replaced by edge $(n'_w \rightleftharpoons w)$. □

Since this means that $(v \Updownarrow w)$ can be ignored, the overlap $(v \Updownarrow w)$ can be removed. This in turn again means that the `E-REDUCE` operation can be applied. This is shown in Figure 4.9b and Figure 4.9c.

The above argument still holds whenever the edges $(n_w \rightleftharpoons w)$ and $(w \rightleftharpoons n'_w)$ have the same capacities a, and the edges $(n_v \rightleftharpoons v)$ and $(v \rightleftharpoons n'_v)$ also have the same capacities b, but $b \neq a$. However if edges $(n_w \rightleftharpoons w)$ and $(w \rightleftharpoons n'_w)$ each have different capacities Lemma 7 does not hold any longer, since replacing the edge will change the value of the solution.

This same trick can be applied to prove that in a stack of structures as given in Figure 4.9a, overlaps can also be removed. An example of such a structure is given in Figure 4.10a. This can be proven using exactly the same steps as before and can only be applied under the same restrictions.

Algorithm 3 `REMOVE`

Input: w: The current vertex under inspection
$\mathbf{O}_v$: The set of old restrictions of the original vertex v
$\mathbf{O}'_v$: The minimal set of needed restrictions for v

```
1: w.visited ← true
2: for ∀o ∈ O_w do
3:     o.conflictCount++
4: end for

5: for n ∈ N_w do
6:     if n.conflictCount > 0 ∨ n.visited then
7:         continue

8:     else if n ∈ O_v then
9:         O'_v ← O'_v ∪ {n}
10:        O_v ← O_v \ n
11:        n.conflictCount++

12:        if O_v = ∅ then
13:            return (∅, O'_v)
14:        end if
15:        continue
16:    end if

17:    (O_v, O'_v) ←REMOVE(n, O_v, O'_v)
18: end for

19: w.visited ← false
20: for ∀o ∈ O_w do
21:    o.conflictCount−−
22: end for
23: return (O_v, O'_v)
```

General case

In the general case, overlaps can be removed whenever the separation of overlapping vertices is already forced by the surrounding environment, just as with `STACK-REMOVE`. In other words, an overlap $(v \Updownarrow w)$ can be safely removed when on all paths connecting v and w there exists a pair of vertices $(p_i \Updownarrow p_j) \in O$. Unfortunately, checking this requires in worst case scenarios an exponential amount of time. A simple BFS or DFS will not suffice since not all paths connecting v and w are traversed. An example implementation of an algorithm that checks what restrictions can be removed, is given in Algorithm 3.

This algorithm is based on a DFS algorithm, without the guarantee that each vertex or edge is only traversed just once. The algorithm uses a boolean value $v.visited\ \forall v \in V$ to check if it is already on the current path under investigation. Besides the boolean for each vertex a counter $v.conflictCount$ is used to check if the vertex has already been blocked by a previous vertex on the path.

Algorithm 4 `INITIALIZE`

Input: v: Vertex of which to check what overlaps can be removed.
$\mathbf{O}_v$: Set of overlaps currently in place for v

1: **for** $o \in \mathbf{O}_v$ **do**
2: $\quad o.conflictCount \leftarrow -1$
3: **end for**
4: $(forget, remainingConflicts) \leftarrow$`REMOVE`$(v, \mathbf{O}_v, \emptyset)$
5: **return** $remainingConflicts$

`REMOVE` should be called as described in Algorithm 4. This algorithm initializes the conflictCounter for every $o \in \mathbf{O}_v$ to -1, so the first call to `REMOVE` will work correctly. In lines 1 through 4 of `REMOVE`, the conflicts of the current vertex are registered. Furthermore, this vertex is marked as visited to ensure no loops are formed. Lines 5 through 18 inspect all the current neighbours of w. On line 6 it is checked if n is still reachable from v when taking in consideration all vertices with overlaps located earlier on the path to n. When there is a vertex on this path that overlaps n, it is not possible to visit n and the algorithm continues with the next neighbour.

Whenever n is reachable from v, it is checked if n is a member of $\mathbf{O}_v$ in lines 8 through 16. If it is, this means that nothing in the environment ensures the separation of v and n. Therefore, n is added to the set of new overlaps for v $\mathbf{O}'_v$. Furthermore, the original conflict from v with n is reactivated. Because this overlap is reactivated, n is no longer reachable through this path since $(v \Updownarrow n)$, so the algorithm continues with the next neighbour of w. If it is possible to visit n, `REMOVE` is called recursively and $\mathbf{O}_v$ and $\mathbf{O}'_v$ are updated accordingly.

Since `REMOVE` is an algorithm that can visit each vertex and edge multiple times, the runtime becomes quite big. Given a graph with max edge-degree E_{deg}, max overlap-degree O_{deg} and $|V|$ vertices. For each individual call to `REMOVE`, $2O_{deg}$ calls are made for lines 3 and 21. Furthermore, the loop on lines 6 through 15 is repeated at most E_{deg} times. Each call to `REMOVE` from line 17 can result in at most $E_{deg} - 1$ new calls to `REMOVE`, since at least one of the neighbours has already been visited. For each of these executions once again at most $E_{deg} - 1$ calls to `REMOVE` can be made. This results in the following recurrence:

$$\begin{aligned} g(0) &= 1 \\ g(n) &= 2O_{deg} + E_{deg} \times g(n-1) \end{aligned}$$

For $n = 1, 2, 3$ the formulas have been expanded below:

$$\begin{aligned} g(1) &= 2O_{deg} + E_{deg} \times g(0) \\ g(2) &= 2O_{deg} + E_{deg} \times (2O_{deg} + E_{deg} \times g(0)) \\ g(2) &= 2O_{deg} + 2E_{deg}O_{deg} + E_{deg}^2 \times g(0) \\ g(3) &= 2O_{deg} + 2E_{deg}O_{deg} + E_{deg}^2 \times (2O_{deg} + E_{deg} \times g(0)) \\ g(3) &= 2O_{deg} + 2E_{deg}O_{deg} + 2E_{deg}^2O_{deg} + E_{deg}^3 \times g(0) \end{aligned}$$

The series $\sum_{i=0}^{n-1} x^i$ is known as the geometric series. When replacing x by E_{deg} and multiplying it by $2O_{deg}$, we get:

$$\sum_{i=0}^{n-1} 2O_{deg}E_{deg}^{i} = 2O_{deg}\sum_{i=0}^{n-1} E_{deg}^{i}$$
$$= 2O_{deg} + 2O_{deg}E_{deg} + \ldots + 2O_{deg}E_{deg}^{n-1}$$

The direct formula for the geometric series is $\frac{1-x^n}{1-x}$. Using this results in the following formula:

$$\sum_{i=0}^{n-1} 2O_{deg}E_{deg}^{i} = 2O_{deg}\frac{1-E_{deg}^{n}}{1-E_{deg}}$$

Adding this to E_{deg}^{n} gives us $O(2x\frac{1-y^n}{1-y} + y^n)$ where $x = O_{deg}$, $y = E_{deg}$ and $n = |V|$.

Since such a worst-case runtime is not usable (especially when you consider running that algorithm for every vertex that has an overlap to check if its overlaps can be removed), it is prudent to limit the search depth at the cost of the number of overlaps that will be removed. Such an algorithm is described in Algorithm 5. This algorithm should be called exactly like Algorithm 3, but with one more parameter. This parameter d is a bound on the path length considered when searching for overlaps that can be removed. Whenever d reaches 0, a simple BFS algorithm suffices for determining whether the overlap has been blocked by the parents or not.

When limiting the depth of the calls of this algorithm, its running time reduces to $O(2x\frac{1-y^d}{1-y} + y^d + y^d DIJKSTRA)$. Instead of a path of maximal length $|V|$, paths of maximal length d are inspected. Furthermore, calls to Dijkstra's algorithm are made to determine the reachability of the overlaps. When the algorithm is called with $d = 1$, it is the same as $1 - REMOVE$ mentioned in Section 4.3.2.

Algorithm 5 `d-REMOVE`

Input: w: The current vertex under inspection
$\mathbf{O}_v$: The set of old restrictions of the original vertex v
$\mathbf{O}'_v$: The minimal set of needed restrictions for v
d: The maximal length of the path to consider

$w.visited \leftarrow$ **true**
for $\forall o \in \mathbf{O}_w$ **do**
    $o.conflictCount++$
end for

if $d = 0$ **then**
    Call Dijkstra from w and block all vertices with $visited = true$ or $conflictCount > 0$

    for $o \in \mathbf{O}_v$ **do**
        if o reachable according to Dijkstra **then**
            $\mathbf{O}'_v \leftarrow \mathbf{O}'_v \cup \{o\}$
            $\mathbf{O}_v \leftarrow \mathbf{O}_v \backslash o$
        end if
    end for
else
    for $n \in \mathbf{N}_w$ **do**
        if $n.conflictCount > 0 \vee n.visited$ **then**
            continue

        else if $n \in \mathbf{O}_v$ **then**
            $\mathbf{O}'_v \leftarrow \mathbf{O}'_v \cup \{n\}$
            $\mathbf{O}_v \leftarrow \mathbf{O}_v \backslash n$
            $n.conflictCount++$

            if $\mathbf{O}_v = \emptyset$ **then**
                return $(\emptyset, \mathbf{O}'_v)$
            end if
            continue
        end if

        $(\mathbf{O}_v, \mathbf{O}'_v) \leftarrow$`REMOVE`$(n, \mathbf{O}_v, \mathbf{O}'_v, d-1)$
    end for
end if

$w.visited \leftarrow$ **false**
for $\forall o \in \mathbf{O}_w$ **do**
    $o.conflictCount--$
end for
return $(\mathbf{O}_v, \mathbf{O}'_v)$

Chapter 5

Heuristic methods

Since we have shown in Chapter 3 that finding a `MIN-T-MLE` is an NP-Hard problem, we will take a look at some heuristic methods. First, we will consider the previous results of Saaltink [40]. In Section 5.2 an implementation of local search will be described. Section 5.3 will contain the implementation details of a genetic algorithm based on Neumann et al. [34]. A heuristic method, based on the observation that all paths, and therefore also all the shortest paths, should be disconnected from a source to the corresponding sink, is described in Section 5.4. In Section 5.5 a heuristic is described that uses the height information of the polygons.

5.1 Previous results

In Saaltink's thesis, he suggested using a heuristic guided by a minimal cost cut algorithm [40]. His heuristic method consists of three phases:

1) Determine the weights for all the edges in the graph;

2) Find a set of candidate transfers/candidate cuts;

3) Iteratively remove candidate cuts that are not needed.

In the first phase of his heuristic, the edges between vertices are weighted according to some properties of the edge. The first property taken into consideration is the number of adjacent edges. The set of adjacent edges of edge $(a \rightleftharpoons b)$ is defined as the set $\{(a \rightleftharpoons x) | (a \rightleftharpoons x) \in E \wedge x \neq b\} \cup \{(b \rightleftharpoons x) | (b \rightleftharpoons x) \in E \wedge x \neq a\})$. If the number of adjacent edges is heigh, the weight of the edge is increased. The second property taken in consideration is the number of overlaps that begin or end in the vertices connected by the edge. When there are conflicts starting and/or ending at both of these edges, the weights were also increased. No further motivation for theses rules was given. The last property is to check if both ends of the edge are two fully connected polygons. A fully connected polygon is a polygon that has the maximal number of neighbours, e.g. a vertex representing a triangle is fully connected when it has three neighbours and a tetragon only if it has four neighbours. If both ends of the edges are fully connected edges, the current weight of this edge is doubled.

In the second phase of the heuristic, a minimal s-t cut algorithm is run for each pair of overlapping vertices. In each run, the minimal set of edges separating the overlapping vertices is found and stored. The next s-t cut is once again run on the original weighted graph obtained at step one. All the edges in the different cut sets together form the set of

candidate transfers $T = \{mincut(a, b) | \forall (a \Updownarrow b) \in O)\}$. It is also shown that using this set T as the cut set will result in a feasible MLE, however this MLE is most likely not optimal.

Saaltink used for his implementation the minimal s-t cut algorithm as defined by Ford and Fulkerson [16]. It was not specified how the graph was transformed to work with this algorithm, however a logical assumption would be that edge $e \in E$ was replaced by two directed arcs between the same vertices and that the overlaps were ignored.

An attempt is made to decrease the number of needed transfers in the last phase of the heuristic. For each transfer it is checked if the layers at each side can be merged, e.g. there does not exist an overlap $o \in O$ such that one end of the overlap is in layer one and ends in layer two. If the two layers can be merged, this is done, thereby reducing the number of transfers.

Another more basic heuristic was also tested, called the Flood-Filling. In this method a random vertex is picked. From this vertex a layer is 'grown' in a Breadth-First manner. Vertices that would overlap the current members of the layer are not visited and therefore not added to the layer. This search is repeated for vertices not part of any layer, untill all vertices are assigned to layers.

Results obtained using Saaltink's original code for both the Flood-Filling and the minimal cut heuristics are shown in Appendix F.

5.2 Local search

Local search is a well known heuristic method based on searching the neighbourhoods[1] of the current solution. For this heuristic method several meta-heuristics exist, including tabu-search [22] and simulated annealing [10, 30]. We implemented these different methods. Short descriptions of these implementations are given in Sections 5.2.2 and 5.2.3. In this section the global implementation will be described. The values of the different parameters used in the experiments will be given in Section 7.3.1 and the found results in Appendix E.

All the neighbourhood operations are defined on components and always return one or more components. Before we move on to the description of the operations, we will first describe how components are represented. A single component has three data members:

members The set of vertices contained within this component;

neighbours The set of vertices that are connected to this component;

conflicts The set of vertices that at least one vertex in **members** overlaps with.

Each vertex that is in the **members** set has a reference to the component it is in. Each solution in the local search assigns every vertex to exactly one component. Furthermore, all components remain valid during the entire process. This means that the intersection of **members** and **conflicts** is always empty. Furthermore, the number of outgoing edges of the component are maintained.

The sum of internal edges of the components is used as scoring function during the local search. The goal of the local search becomes finding the set of components with the biggest

[1] A neighbour of a solution x is a different solution obtained from x by some mutation. All these different mutations together form the neighbourhood of x.

Algorithm 6 Pseudocode for the `MERGE` operation

Input: a: A component;
b: A component.

1: **if** $a.members \cap b.conflicts = \emptyset$ **then**
2: **return** t-`MERGE`(a, b)
3: **else**
4: **return** `REDIST`(a, b)
5: **end if**

Algorithm 7 Pseudocode for the t-`MERGE` operation

Input: a: A component;
b: A component.

1: $c.members \leftarrow a.members \cup b.members$
2: $c.neighbours \leftarrow (a.neighbours \cup b.neighbours) - c.members$
3: $c.conflicts \leftarrow a.conflicts \cup b.conflicts$
4: **return** c

number of internal edges. When this set of components is found, we also have the minimal cut set. The three operations used by the different local search implementations are listed below and explained in more detail later on.

`MERGE` Combine the members of two neighbouring components. If A has no conflict with B, a trivial merge is performed. Otherwise, an instance of min-cut max-flow is solved for these two components only;

`MOVE` Move x members of component A to neighbouring component B in such a way that the components remain feasible (conflicting vertices cannot become member of the same set);

`SPLIT` Split a component in two new components such that the decrease in profit is minimal (solve the global minimal cut problem for a single component).

The `MERGE` operation is described in Algorithm 6 through 8. Whenever the `MERGE` operation is called, it checks whether it can use a trivial merge (t-`MERGE`, Algorithm 7). When this is not the case, instead of actually merging the two components, it will attempt to redistribute the components using `REDIST` (Algorithm 8). The t-`MERGE` operation is applied whenever the components do not conflict and simply consists of performing several unions and intersections on the members, neighbours and conflicts of the components.

The `REDIST` algorithm tries to redistribute the vertices contained within the two components in such a way that the size of the cut separating the two components becomes minimal, but it does not guarantee optimality. It does this by first creating the sub-graph G' induced by the members of the components a and b. Secondly, a super source s and a super sink t are added to the subgraph. Edges are added between s and the members of component a that overlap members of b, and members of component b that overlap members of a are connected with edges to t. The capacities of these edges are set to infinity and the weight to one, ensuring that these edges will never be cut.

Algorithm 8 Pseudocode for the redistribution of the components

Input: a: A component;
b: A component.

1: $c \leftarrow a.members \cup b.members$
2: $G' \leftarrow$ graph induced by vertices c

3: $s \leftarrow a.conflicts \cap b.members$
4: $t \leftarrow b.conflicts \cap a.members$

5: $(a, b) \leftarrow$s-t-cut(s, t, G')
6: **return** a, b

This method does not guarantee that the overlapping vertices in a and b are separated optimally. One way to accomplish this would to solve an instance of `MULTICUT` which is, as we know, an NP-Hard problem. Another method would be to try all possible permutations of connecting the overlapping vertices to either the source s or the sink t. However, this would require $2^{|O'|}$ minimal s-t cut computations, where O' is the set of overlaps in G'.

An important thing to note is that the solution after a `MERGE` operation will always have at most the same number of cuts as before `MERGE` was applied. Whenever a trivial merge can be performed, it is easy to understand that the number of cuts will decrease. Whenever the `REDIST`-algorithm is called, a set of edges that is needed to separate all overlapping vertices is found. Since this operation does not influence the number of cuts going to the remainder of the graph, this number will not change. Since the s-t-cut algorithm finds the minimal number of edges separating component a from component b when the overlapping vertices are fixed to their currenct components, it will always decrease the minimal number of cuts needed unless that cut was already minimal.

The `MOVE` operation is described in Algorithm 9. It takes two neighbouring, possibly overlapping components $from$ and to and attempts to move x members of $from$ to to. This operation allows for both increase and decrease of the number of cuts in the current solution. It is a very important operator since it can move the current solution away from a local optimum.

The `SPLIT` operation is an implementation of a min-cut algorithm, in this case the algorithm as described by Brinkmeier in [8]. This algorithm finds a global min-cut in an undirected weighted graph in $O(\delta_G|V|^2)$ time. Here δ_G is the minimal vertex degree in G, or $\delta_G = \min_{v \in V}(deg(v))$ where $deg(v)$ is the sum of the edge weights of v. This algorithm takes as input one single component and returns two components. Therefore, it will always increase the number of cuts needed. The reason for using this operator is the same as for using the `MOVE` operator, it allows for getting out of a local optimum.

The experiments done with local search are described in Section 7.3.1 and the results are given in Appendix E.

5.2.1 Hillclimbing

Hillclimbing [39] is the most basic version of local search. When we use this heuristic, a randomly selected neighbourhood operation is applied to the current solution. Whenever

Algorithm 9 Pseudocode for the `MOVE` operation.

Input: $from$: A component from which vertices are moved to to.
to: A component to which vertices are moved from $from$.
n: The number of attempted moves.

```
1:  adjacentNeighbours ← to.neighbours ∩ from.members
2:  moves ← 0
3:  while moves < n do
4:      adjacentNeighbours ← adjacentNeighbours\to.conflicts
5:      if |adjacentNeighbours| = 0 then
6:          break
7:      end if
8:      i ← random number between 0 and |adjacentNeighbours| − 1
9:      w ← adjacentNeighbours.at(i)
10:     adjacentNeighbours ← adjacentNeighbours\w
11:     to.members ← to.members ∪ {w}
12:     from.members ← from.members\w
13:     to.neighbours ← (to.neighbours ∪ w.neighbours) ∩ to.members
14:     for n ∈ from.neighbours do
15:         isNeighbour ← false
16:         for nn ∈ n.neighbours do
17:             if nn ∈ from.members then
18:                 isNeighbour ← true
19:                 break
20:             end if
21:             if ¬isNeighbour then
22:                 from.neighbours ← from.neighbours\n
23:             end if
24:         end for
25:     end for
26:     from.neighbours ← from.neighbours ∪ {w}
27:     moves ← moves + 1
28: end while
```

a newly obtained solution is an improvement compared to the current solution, it becomes the new current solution. When an improvement has not been found for a predetermined consecutive number of iterations, the algorithm terminates.

5.2.2 Local search and simulated annealing

Simulated annealing [10, 30] is a form of local search. Instead of rejecting a solution that is not an improvement to the current solution, it is rejected with a certain probability. This

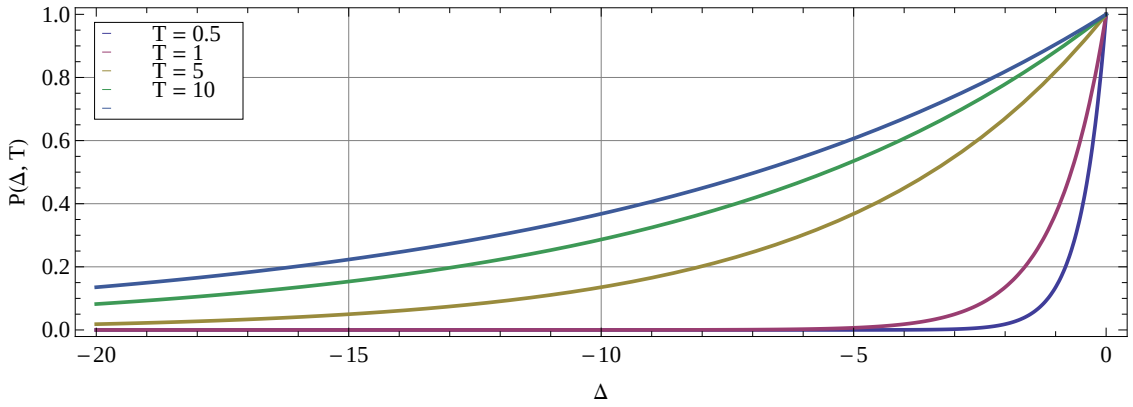


Figure 5.1: A plot showing the chance a new, worse solution is accepted for different temperatures and different decreases in solution value.

probability is determined using the following functions:

$$P(\Delta, T) = e^{\Delta/T}$$

In this formula, Δ is the increase in profit, which is in our case the increase in internal edges for the new solution. T is the current temperature of the simulated annealing instance. Whenever there is a decrease in profit, $P(\Delta, T)$ returns a value in the range $[0, 1]$. This value is used in conjunction with a random generated number r in the same range to determine if the new solution is accepted. Whenever $r < P(\Delta, T)$, the new solution is accepted. When $r \geq P(\Delta, T)$, the new solution is rejected. Whenever Δ has a positive value the new solution is always accepted. A plot showing different acceptance ratios for different temperatures is shown in Figure 5.1.

The chances of accepting new solutions with negative Δ should decrease over time. This is accomplished by lowering the temperature T. This change of temperature is dependent on the cooldown factor C. During the search, the best ever seen solution is stored.

5.2.3 Tabu search

In tabu search [22], a tabulist is maintained. This list contains a history of the last n solutions. The solutions are encoded by the cut edges for that particular solution. Whenever a new solution is generated that is in the tabulist, it is disregarded. The new solution is picked out of a neighbour list of a predetermined size. This number of predetermined neighbours, called the neighbourhood-size ns, can be changed before a tabu search is started. The neighbour list is generated each iteration by generating ns random neighbours using the neighbourhood operations. The best neighbour that is not in the tabulist is the neighbour that becomes the new current solution.

5.3 Genetic algorithm

Genetic algorithms belong to another class of methods used to solve computational intensive problems [39]. The general idea of genetic algorithms is to simulate the evolutionary process.

This is done by encoding a problem using a genotype. This genotype encodes a phenotype of the problem. This phenotype is the actual solution encoded using the genotype. Using the phenotype the fitness of the genotype is determined using a fitness function.

Another characteristic of genetic algorithms is that a population of solutions is maintained. From this population descendants are generated using mutations and/or recombination. This process continues until some stopping criterion is met.

5.3.1 Solving the MIN-T-MLE problem using genetic algorithms

A genetic algorithm has been applied on the MULTICUT problem by Neumann et al. [34]. As a genotype a simple bitstring $x \in \{0,1\}^{|E|}$ was chosen, where a 1 means that a certain edge is cut. The bitstrings were mutated by flipping each individual bit with probability $\frac{1}{|E|}$. No recombination operators were applied.

The phenotype is assessed using two criteria, the number of cuts made (the number of bits set to 1 in the genotype) and the summation of all residual s-t flows for all the pairs of overlapping vertices. Newly generated genotypes were introduced in the population whenever it was not dominated by any other gene currently in the population. Furthermore, whenever a member of the population becomes dominated, it is removed from the population.

The reason for picking a component wise fitness function is explained in a previous paper written by Neumann et al. [35]. Neumann et al. proved that a single-objective algorithm searching for a s-t cut in a graph would often get stuck in local minimum. Using the previously described method with two objectives to be optimized resulted in an optimal result in expected polynomial time for the s-t cut problem. Since MULTICUT is harder, using a single objective algorithm would be useless according to Nuemann et al.

Using this method, infeasible solutions can remain in the population as long as the number of cuts is small enough. However, if a solution with no residual flow was found, from that point on there would always be a feasible solution in the population, since only dominated solutions are removed. This feasible solution with residual flow 0 can only be dominated by another feasible solution with a residual flow of 0.

5.3.2 Adding recombination

Neumann et al. [34] used their genetic algorithm on unrestricted graphs. However, the graphs under consideration in this thesis have some special restrictions. Most of these restrictions are already discussed on page 18. One more restriction on the graphs is that the pairs of vertices that have an overlap are located in approximately the same location in the xz-plane. This observation is used when creating the recombinator. Experiments comparing the original algorithm as described by Neumann et al. [34] and the algorithm using recombination are described in Section 7.3.2. The results can be found in Appendix D.

The recombinator places an xz-grid over the graph, as is shown in Figure 5.2. Whenever two genotypes are recombined, consecutive blocks from one gene are swapped with the same blocks in the grid of the other genotype. This might enable merging good regions with each other. The size of the grid is a fixed parameter and does not change during the whole run of the algorithm.

To speed up the recombination operation, all edges are assigned to a box in the grid before the genetic algorithm starts. The location of an edge is calculated by taking the average of

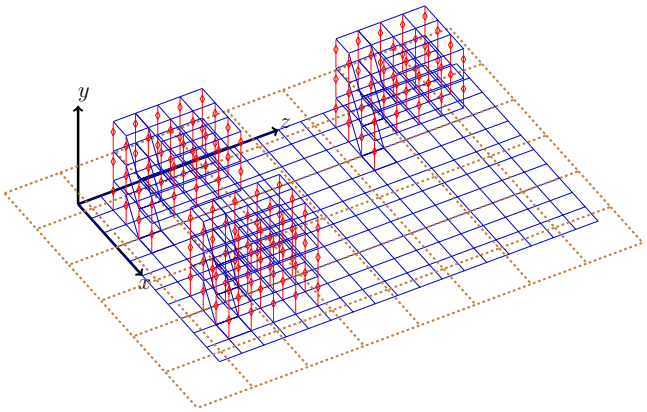


Figure 5.2: An example P-Graph. The brown-dashed lines depicts the grid that is used for the recombination operation for the genetic algorithm.

the centres of the two vertices it belongs to. This enables for fast retrieval of the edges in a specific area on the grid.

5.4 Shortest Path Heuristic

The shortest path heuristic is based on the observation that all paths connecting two overlapping vertices should be cut, as was previously discussed in Section 4.3. For this reason, all the shortest paths connecting each overlapping pair of vertices should be cut too. Finding shortest paths in a graph is a topic that has been studied extensively. The shortest path algorithm that will be used here is the algorithm as described by Johnson [12, 28]. This algorithm finds the shortest path between all pairs of vertices in $O(|V||E| \log |V|)$ time when it is implemented using binary heaps. This algorithm was used because it outperforms other graph-algorithms on sparse graphs, which the **P**-Graph is. The implementation of Johnson's algorithm in the BGL (Boost Graph Library, [1]) returns a matrix with the distances between all the vertex pairs. Using this matrix, all paths can be traversed. While traversing a path using the matrix, it should be the case that whenever there exists more than one option for continuing a path, the choice of which option to take should be made at random.

Using a priority queue the most often traversed edges can be easily found. Whenever an edge is removed, the paths going through this vertex should be removed. Afterwards, new shortest paths should be searched for using Dijkstra's algorithm. This process continues until no paths between source-sink pairs remain. During this entire algorithm all edges have length one. The results for this algorithm can be found in Appendix B.

5.5 Height Heuristic

The last heuristic method created is the Height Heuristic (`HH`). This heuristic consists out of three distinct phases. During the `CLUSTER` phase, polygons are clustered using height information. Using the height clustered polygons, connected components are created. This process is described in Algorithm 10. It is assumed that for every polygon the y coordinate is known and accessable as a member of the polygon. The function AvgHeight(x) will return

the average height of the polygons in set x.

On lines 1 through 8, a very simple clustering is made. Polygons that are exactly the same height, are grouped together. On lines 9 through 41, groups of polygons that are neighbouring in height are merged. This process starts with first clustering groups with a small height difference. When there is no conflict between these clusters, they are merged. This repeats until no clusters with sufficiently small height difference exist. When this happens, the range in which groups can be merged is multiplied by ten. When all remaining clusters are close enough to each other, but cannot be merged because there is a conflict, `CLUSTER` terminates. The resulting clustering found in priority queue *out* is converted into connected components.

In the `MERGE` phase, non-conflicting neighbouring components are merged. During the last phase, called `REDIST`, the neighbouring conflicting components are redistributed. For each component an attempt is made to redistribute its members with each of its neighbouring components. When a component changes during this process, this is recorded. If no component changes, this last phase is completed. If at least one of the components changed, the process is repeated for all components that did change. The `MERGE` and `REDIST` operations are the same as the `t-MERGE` and `REDIST` operations described in Section 5.2. The results for the experiments done with this algorithm are given in Appendix C.

Algorithm 10 Pseudocode for the `CLUSTER` operation.

Input: **P**: The corresponding polygonal environment of G.

```
    // Cluster polygons at exactly the same height.
 1: m ←new multi_map
 2: for P ∈P do
 3:     m.insert(P.y, P)
 4: end for
    // Store groups of polygons ordered on height.
 5: in ← new PriorityQueue; out ← new PriorityQueue
 6: for bucket ∈ m do
 7:     in.insert( bucket.key, bucket.values )
 8: end for
 9: swap ← out; out ← in; in ← swap
10: range = ϵ
11: notOnlyConflict ← TRUE
12: while notOnlyConflict and out.size() > 1 do
13:     notOnlyConflict ← FALSE
14:     changed ← FALSE
15:     previous ← out.pop()
16:     while out.size() > 0 do
17:         prevHeight ← AvgHeight(previous)
18:         prevConflicts ← Conflicts(previous)
19:         prevMembers ← previous
20:         current ← out.pop()
21:         curHeight ← AvgHeight(current)
22:         curConflicts ← Conflicts(current)
23:         curMembers ← current
24:         conflict ← curMembers ∩ prevConflicts ≠ ∅
25:         if |curHeight − prevHeight| > range or conflict then
26:             if ¬conflict then
27:                 notOnlyConflict ← TRUE
28:             end if
29:             in.insert(prevHeight, previous)
30:             previous ← current
31:         else
32:             changed ← TRUE
33:             previous ← prevMembers ∪ curMembers
34:         end if
35:     end while
36:     in.push(AvgHeight(previous), previous)
37:     if ¬ changed and notOnlyConflict then
38:         range ← range ∗10
39:     end if
40:     swap ← out; out ← in; in ← swap
41: end while
```

Chapter 6

Linear programming

In this chapter we consider the application of linear programming techniques for finding good results for the MIN-T-MLE-cut problem. Just as with local search, the goal will be to find non-overlapping, feasible connected components rather than finding the set of cutting edges. As argued before in Section 5.2 this set of connected components will yield the same results as just searching for a set of cutting edges.

In the (I)LP framework, a connected component c_i will be described using two parameters. L_i is a vector of length $|V|$, with $L_{i,v} = 0$ whenever vertex v is not in this connected component. Whenever vertex v is in connected component L_i, $L_{i,v}$ will be 1. The other parameter is C_i, the number of internal edges of such a connected component. Whenever the set of all possible, feasible connected components is given and has size n, the ILP for finding a MIN-T-MLE can be formulated in the following fashion:

Maximize: $\sum_{i=1}^{n} x_i C_i$

Subject to:

$$\sum_{i=1}^{n} x_i L_{i,v} = 1 \qquad \forall v \in V \qquad (1)$$

$$x_i \in \{0, 1\} \qquad 1 \leq i \leq n \qquad (2)$$

Equation (1) guarantees that a vertex will be in exactly one connected component, equation (2) ensures that a component is selected completely or not at all. The LP-relaxation of the above problem is:

Maximize: $\sum_{i=1}^{n} x_i C_i$

Subject to:

$$\sum_{i=1}^{n} x_i L_{i,v} = 1 \qquad \forall v \in V \qquad (3)$$

$$x_i \geq 0 \qquad 1 \leq i \leq n \qquad (4)$$

The variables x_i are no longer binary variables. The upper bound of these variables is not given, however the upper bound is still enforced by constraint (3).

6.1 Column generation

To solve the given LP, it would be necessary to determine all $2^{|V|} - 1$ possible components in the graph (worst case scenario). Therefore, we will employ a technique called column generation. This technique allows us to start searching for a solution while not knowing all possible connected components. It also allows for searching for new components that will increase the value of the current objective whenever the current solution is not optimal. Whenever the current solution is optimal, no new components will be found.

Suppose that a new component has been generated and it has corresponding L_0, C_0 and decision variable x_0. When x_0 gets fixed to a very small ϵ, the following is obtained:

Maximize: $x_0 C_0 + \sum_{i=1}^{n} x_i C_i$

Subject to:

$$\sum_{i=1}^{n} x_i L_{i,v} = 1 - x_0 L_{0,v} \qquad \forall v \in V \qquad (5)$$

$$x_i \geq 0 \qquad 1 \leq i \leq n \qquad (6)$$

$$x_0 = \epsilon$$

Since ϵ is fixed to be an arbitrary small value, we do not have to solve the entire system to see if the introduction of layer L_0 would lead to a possible increase in profit. The direct increase in profit is given by $x_0 C_0 = \epsilon C_0$. Unfortunately the introduction of component 0 also takes up some of the capacity of constraint (5) used by the previous solution. The cost for changing the capacity of a constraint is known as the shadow price. For instance, decreasing the capacity of constraint (5) for vertex v would cost $x_0 L_{0,v} \pi_v$. Here π_v is the shadow price for the constraint corresponding to vertex v. The value of π_v is known. Putting these two things together, the following change in profit can be found:

$$\begin{aligned} I &= x_0 C_0 - x_0 \sum_{v \in V} \pi_v L_{0,v} \\ &= \epsilon C_0 - \epsilon \sum_{v \in V} \pi_v L_{0,v} \end{aligned}$$

The reduced cost is defined to be the unit change in costs, which is $\frac{I}{x_0} = \frac{I}{\epsilon} = C_0 - \sum_{v \in V} \pi_v L_{0,v} = RC$. When the reduced cost is negative for component 0, we know that introducing this component in the current solution will not increase the current profit. Only when the reduced costs are positive (which would mean that the value C_0 is bigger than $\sum_{v \in V} \pi_v L_{0,v}$), it is interesting to introduce this new found component. Finding such a component with positive RC is called the pricing problem.

6.2 Solving the pricing problem

In this section we will take a closer look at solving the pricing problem. In Section 6.2.1 the pricing problem will be solved to optimality using another ILP, whereas in Section 6.2.2 the pricing problem will be solved using a local search method. Both methods attempt to solve the same pricing problem, which is to maximize $C_0 - \sum_{v \in V} \pi_v L_{0,v}$. Here C_0 is the number of internal edges of a connected component, π_v is the shadow price as defined before and $L_{0,v}$ is 1 whenever v is in component 0.

The constraints that the component has to satisfy are as before. First of all, no single pair of vertices in a component may overlap. Second of all, the vertices in a component should form a connected component.

6.2.1 Solving the pricing problem as another ILP

Making sure that no overlapping vertices are selected in one component is a simple constraint in the ILP. Assuming that for each vertex v there exists a decision variable $y_v \in \{0, 1\}$, where 0 means the vertex is not in the component, an overlap can be encoded as:

$$y_v + y_w \leq 1$$

When vertex v is not selected, selecting w will still satisfy the constraint. The same holds when w is not selected and v is. Only when both vertices are selected, the constraint will not be satisfied.

To make sure the selected vertices will form one connected component, we use the concept of legal flow as defined in for example Cormen et al. [12]. A legal flow has to have three properties:

Flow conservation: For all $v \in V$ the incoming flow should equal the outgoing flow ($\sum_{w \in V} f_{v,w} = 0$), with exception of the sources and sinks in the network;

Skew symmetry: For all $v, w \in V$, the flow $f_{v,w}$ should be $-f_{w,v}$;

Capacity constraints: For all $v, w \in V$ the flow $f_{v,w}$ should be at most the capacity $c_{v,w}$ of that edge.

Assuming it is known in advance that one vertex $s \in V$ will be part of the optimal connected component, it is possible to find the optimal connected component, that is, the connected component with the highest reduced costs containing s. This is achieved with the following steps.

First, each vertex $v \in V \backslash \{s\}$ is given a demand of 1 and all edges a capacity of $x_{v,w}|V|$. Here $x_{v,w}$ is a decision variable corresponding to an edge $(v \rightleftharpoons w) \in E$. $x_{v,w}$ has the value 1 whenerver the edge is part of the connected component and 0 the edge is not in the component. The vertex s, of which is assumed it is part of the optimal connected component, is given an infinite supply.

Next, a new vertex s' is added. This vertex has no demand and an infinite supply. Edges of capacity 1 are added from s' to all $v \in V \backslash \{s\}$.

Finally, four more constraints are added. The first constraint guarantees that only one of two conflicting vertices can be selected. The second constraint ensures that $x_{v,w}$ can only be positive whenever y_v and y_w are both positive. The next constraint fixes the flow through the edges $(v \rightleftharpoons w) \in E$ to 0 whenever $x_{v,w} = 0$. The last constraint fixes the flow through edges $(s' \rightleftharpoons v)$, $v \in V \backslash \{s\}$ to 0 whenever $y_v = 1$.

(a) (b)

Figure 6.1: An example of how a graph needs to be modified to solve the pricing problem. In (a) the original graph is given. Subfigure (b) shows a graph where vertex D is chosen as a source and s' is added to the graph. Along the edges are the capacities of each edge, where $c(b) = 1 - y_b$ and $c(a, b) = |V|x_{a,b}$.

An example of such an ILP is shown below. An example of the graph that corresponds to the ILP is given in Figure 6.1.

Maximize: $\sum_{(v \rightleftharpoons w) \in E} x_{v,w} - \sum_{v \in V} y_v \pi_v$

Subject to:

$$\sum_{(v \rightleftharpoons w) \in E \setminus \{v,s'\}} x_{v,w} \leq y_v |E_v| \qquad \forall v \in V \qquad (7)$$

$$y_v + y_w \leq 1 \qquad \forall (v \Updownarrow w) \in O \qquad (8)$$

$$f_{v,w} + f_{w,v} = 0 \qquad \forall (v \rightleftharpoons w) \in E \qquad (9)$$

$$f_{s',v} + \sum_{(w \rightleftharpoons v) \in E} f_{w,v} = 1 \qquad \forall v \in V \setminus \{s, s'\} \qquad (10)$$

$$f_{v,w} \leq |V| x_{v,w} \qquad \forall (v \rightleftharpoons w) \in E \qquad (11)$$

$$f_{s',v} + y_v \leq 1 \qquad \forall v \in V \setminus \{s, s'\} \qquad (12)$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \setminus \{s'\} \qquad (13)$$

$$x_{v,w} \in \{0, 1\} \qquad \forall (v \rightleftharpoons w) \in E \qquad (14)$$

In this ILP, $f_{x,y}$ is the flow between vertices x and y. The objective function is $\sum_{(v \rightleftharpoons w) \in E} x_{v,w} - \sum_{v \in V} y_v \pi_v = C_0 - \sum_{v \in V} y_v \pi_v$, which is the reduced cost that we want to maximize. Constraint (10) is the demand for the vertices and constraint (11) and (12) encode the edge capacities. Constraint (7) is the constraint that ensures that an edge $(v \rightleftharpoons w)$ can only be selected whenever both $y_v = 1$ and $y_w = 1$. Constraint (8) encodes the overlaps.

It is easy to see that at no moment in time two overlapping vertices $(v \Updownarrow w) \in O$ will be selected by the ILP given above. This would mean that both $y_v = 1$ and $y_w = 1$, which would violate constraint (8). Furthermore, when $y_v = 0$ for $v \in V \setminus \{s, s'\}$, its demand can be satisfied by making $f_{s',v} = 1$, since $f_{s',v} + y_v = 1 + 0 \leq 1$. This also happens to be the only possibility to satify v its demand when $y_v = 0$, since constraint (7) forces all $x_{v,w} = 0$. This forces all flow to v through any edge other than through $(s' \rightleftharpoons v)$ to 0 by constraint (11).

An example of what happens when $y_v = 1$, is given in Figure 6.2. At the beginning, all vertices except vertex s have $y_v = 0$, as in (a). In (b) $y_A = 1$. Because of this, the flow $f_{s',A} = 0$. The only possibility to satisfy the demand of A is by increasing the flow through

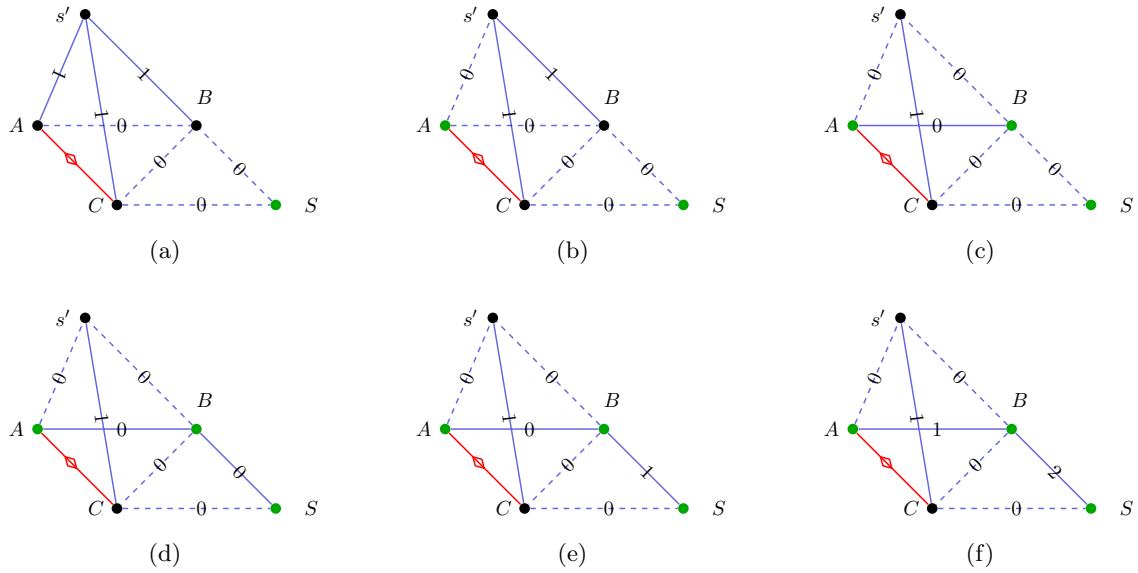


Figure 6.2: An example of how the vertices with $y_v = 1$ have to form a connected component. In the images black vertices have $y_v = 0$. Dashed edges have $x_{v,w} = 0$. The number along the edges is the flow through the edges.

$f_{B,A}$ to 1. However, there is only flow possible through an edge when both of its vertices are selected because of constraint (7). This situation is shown in (c). In this situation, both A and B do not satisfy their demand. The only possible routes to satisfy the demands of A and B is by increasing the flow through $f_{C,B}$ or $f_{s,B}$. Lets first consider increasing the flow through $f_{C,B}$.

To increase the flow through $f_{C,B}$, first $x_{B,C}$ has to become equal to 1. However, this can never happen because this would violate constraint (8). Therefore, the flow through $f_{s,B}$ has to be increased. Since s is already selected, this can be done immediately and we end up in (d). Since vertex s has no demand and infinite supply, it can satisfy the demand of both B and A. This is shown in (e) and (f).

As can be seen from this example, whenever a vertex v is part of the component, the only possible way to satisfy its demand is by a flow originating in s. Therefore, there needs to exist at least one path from v to s for which all vertices p on that path have $y_p = 1$. Therefore, the component found using this ILP will form a connected component.

Unfortunately, it is not possible to determine beforehand if a certain vertex s will be part of a connected component with positive reduced cost. Therefore, it is necessary to solve the ILP once for each vertex selected to be s. Fortunately the lower bound of the Branch-and-Bound process can be increased depending on the previous result. In the first iteration, the lower bound of Branch-and-Bound can be set to $\max_{v \in V} -\pi_v$. During the next iteration the initial lower bound can be set to the final lower bound of the previously solved ILP using Branch-and-Bound. Whenever a better connected component has been found, the lower bound will be equal to the value of the objective function corresponding to that component. When no better connected component has been found, the lower bound has not been changed during this run of Branch-and-Bound. Using this method can result in faster termination

of the Branch-and-Bound algorithm. However, it is not necessary to search for the optimal component, it is only necessary to search for a component with positive reduced cost.

6.2.2 Solving the pricing problem as another local search

The pricing problem can be solved with a local search similar to the one described in Chapter 5. It is still necessary to find connected components that do not contain overlapping vertices, however two things changed. Besides the number of internal edges for each component, the sum of all π_v of the vertices within the components does also matter. Furthermore, we are only interested in finding one single component, instead of a set of non-overlapping components covering the entire graph. The neighbourhood operators of this local search are:

`ADD` Add a vertex that neighbours a vertex already in the component and does not overlap any vertex already in the component;

`REMOVE` Remove a vertex that has at least one neighbouring vertex not in the component from the component. If this makes the component disconnected, take the component with the highest objective score as the new component;

`SWITCH` Add a vertex that neighbours a vertex already in the component and does overlap a vertex already in the component. All the vertices already in the component that have an overlap with the newly added component should be removed. If this makes the component disconnected, take the component with the highest objective score as the new component.

For the `ADD` operation, checking if new conflicts occur can be done in $O(O_{deg})$ time, where O_{deg} is the maximum number of vertices one vertex overlaps with. When a `REMOVE` operation is performed, it is possible to check in amortized $O(\log_2^2 |V|)$ time if the component has become disconnected using the method described by Thorup et al. [24]. The same goes for the `SWITCH` operation.

6.3 Branching to find the optimal solution

While solving the LP-relaxation of the `MIN-T-MLE` problem described on page 47 with the use of column generation, it might be the case that an integral solution is found. Whenever this happens an optimal feasible solution for the ILP has been found. However, more likely than not, a fractional solution will be found. Whenever this happens the usual approach is to solve the problem using Branch-and-Bound [31].

In the Branch-and-Bound algorithm a local upper-bound and a global lower-bound are maintained. The lower-bound is the solution value of the best known integral solution found thus far.

Whenever a solution is fractional and the solution value is higher than the lower-bound, the algorithm branches on one of the fractional variables. In one of the branches one of the fractional variables is fixed to 0 and in the other branch to 1. The upper-bound of these two branches will be set to the value of the solution as found by the current fractional solution.

Whenever the current branch of the search tree does not result in a feasible solution for the LP relaxation, no further action is taken in the current branch. If it is the case that

a integral solution is found, the global lower-bound is fixed to this value whenever the new value is an improvement over the old value. The search tree can be formed in several ways, for example breadth-first or depth-first. As long as only branches of the tree are cut of whenever it is guaranteed no better solution can be found in this branch of the search tree, an optimal solution for the current set of available columns will be found.

Unfortunately, this method will only result in an optimal solution whenever all the columns are known. Since this is not the case (since we are using a pricing problem to find new columns), other measures need to be taken. One way to tackle this is to generate more columns than are needed for solving the initial LP-relaxation. How this can be used is described in Section 6.3.1. This does not guarantee an optimal solution.

Another method that will result in an optimal solution is described in Section 6.3.2.

6.3.1 More columns

When solving the ILP using the columns generated during the column generation phase, we do not have all possible columns available. As a result it is likely that a sub-optimal solution is found. To decrease the gap between the found solution and the actual optimal solution, we will generated more columns during the column generation than is strictly necessary. This will be done during the pricing phase. Whenever a column is generated during the pricing phase, we will ensure that this column will not be generated again. This process will be repeated a pre-determined number of times. Using all these generated columns, the ILP will be solved.

These columns can be found using several methods. When using the LS Pricer, it is possible to store all feasible components with positive reduced cost. When searching for the component with the highest reduced cost using the ILP Pricer, it is possible to list all feasible components with positive reduced cost and add all of these columns. When the ILP Pricer is used and it is terminated as soon as a feasible component with positive reduced cost is found, we can make little changes to the found solution, for example by using the LS operations as described in Section 6.2.2. The method that was used is to store all columns encountered during Branch-and-Price, as described in Section 6.4.4. This is the method that was used during the experiments.

6.3.2 Branch-and-Price

Another way to circumvent this problem is by using Branch-and-Price. This methodology is an extension to Branch-and-Bound and it goes very well with Column Generation [4]. In Branch-and-Price, the LP-relaxation is solved as usual. Whenever the found solution is fractional, it once again becomes necessary to branch. Instead of branching on the x_i variables, we branch on edges. In one branch a certain edge will be forced to be cut, while in the other branch the edge is forced not to be cut. This change is also reflected in the LP-relaxation. How this is implemented is described in Section 6.4.

When fixing an edge to be either cut or not, this has two very important side effects. Firstly, it ensures that during the pricing phase no previously blocked layer is regenerated. Secondly, it reduces the search space. Since certain edges get fixed to either be a cutting edge or not, no new layers can be generated that contain both the neighbours of this edge. For this reason, searching for columns during the pricing phase does not get more difficult.

Simply branching on the x_i variables will not have the desired result. This will forbid the current version of the connected component x_i, but no guaratnee is given that it will never be generated once again in a later stage.

Unfortunately, it also comes at a cost. Branch-and-Bound itself is a very flexible methodology, usually no extra implementation is needed to get it running. Branching rules for Branch-and-Price are unfortunately problem specific. The branching rule that will be used in this Branch-and-Price solution will branch on one of the edges leaving a layer selected in the unrestricted linear program that was only selected partially. We know that such a layer must exist, otherwise there would be no need for branching.

6.4 Implementation

The models described in the previous sections were implemented in the ZIB Optimization suite, using SCIP [2] and SoPlex [45]. The ZIB Optimization suite is developed at the Konrad-Zuse-Zentrum für Informationstechnik in Berlin. The main reasons for using this framework is that it is very flexible and it has Branch-and-Price capabilities. In the following sections the different implemented parts are described.

6.4.1 Branching

In SCIP, it is possible to implement custom branching rules. The branching rules have different entry points. The first entrypoint, `scip_execlp`, is the callback for normal branching. A branching is 'normal' whenever there are no new columns with positive reduced cost ($\sum_{v \in V} \sum_{w \in V \setminus \{v,s'\}} x_{v,w} - y_v \pi_v > 0$) and the found solution is not integral.

However, whenever a pricer is used that does not guarantee no columns with positive reduced cost remain, it can be the case that there is an integral solution to the LP while this is not the optimal solution for that particular branch. Whenever such a pricer is used, this can be signalled to SCIP. How this is done is described in Section 6.4.2. Instead of calling the `scip_execlp` entry point to the branching rule, the `scip_execps` entry point is used. In SCIP-terminology, branching using the `scip_execps` entry point is called a pseudo-branching. Instead of branching on a fractional variable, one is supposed to branch on variables that are not fixed in the current solution.

For both types of branching, the same branching rule described in Section 6.3 is used. This means an edge belonging to a fractional (or not fixed) component is selected. On one branch this edge is cut while in the other branch the vertices belonging to this edge are contracted. Branching in such a fashion also influences the local upper bound of that branch. Whenever an edge is cut, or a new cut is forced when two vertices get merged, the upper bound is adapted appropriately.

6.4.2 Pricing

A pricer implemented in SCIP usually needs at least two entry points. The first entry point, `scip_redcost` is used for solving the pricing problem for each node in the search tree. From this entry point, either the local search algorithm or the ILP pricer is called. First an attempt is made to generate new columns using the local search algorithm. Only when there are no new columns found using this method the ILP pricer is called. The reason for this is that,

although the ILP pricer is guaranteed to find new columns with positive reduced cost if any exist, it is really slow. For smaller environments, pricing using the ILP pricer takes several minutes, while searching for columns in the bigger environments (such as max described in Section 7.1) can take several hours. Because the ILP pricer takes so much time for finding a feasible solution with positive reduced cost, sometimes the ILP pricer is even skipped when the local search pricer does not find new columns. SCIP supports this feature, called early branching, by not updating the upper bound of that particular branch. Of course this has the negative side-effect that too much branching might occur.

The second entry point, `scip_farkas` should be implemented for handling the situations, where branching results in an infeasible LP solution. This can happen whenever the solution actually is infeasible, in which case the Farkas pricing should not find any new columns to resolve the problem, or whenever the solution becomes infeasible because the needed columns were previously not generated. In the second case the Farkas pricing should find enough columns to resolve this issue. This entry point is however not implemented, because of the heuristic method that is used and described in the next section.

6.4.3 More heuristics

Besides a heuristic pricer, there is also a heuristic implemented that is executed after each branching but before a branch is cut off. This is done first and foremost to check if there still exists a feasible solution for the `MIN-T-MLE` problem after this branching. It can be the case that the columns generated up and untill now can not form a feasible solution after the branching. In Section 5.2, very fast methods have been created that will always find a feasible solution for the `MIN-T-MLE` problem whenever one is available. These local searches are run on the transformed graph, which includes all the restrictions made by the branchings. A nice side effect is that the heuristic method might find new, better primal solutions, thereby tightening the global lower bound for the search tree.

6.4.4 More columns

Despite all the effort, finding an optimal solution might still take a very long time, mainly because of the ILP pricer. It is needed to guarantee that the current solution is optimal. However, when an optimal solution cannot be found in a reasonable amount of time, all the generated columns are stored in a file. These columns are used as if these were all possible columns in a separate branch and bound run, not using any of the methods described before.

Chapter 7

Experiments

In this chapter the experiments and experimental setup will be described. The algorithms described in Saaltink's thesis will be tested on the same environments, so comparisons can be made. The results will be described in Chapter 8. The number of times an experiment will be repeated will be stated at each individual experiment. All our experiments were run on Intel Xeon E5420@2.50GHZ machines with 4GB of RAM. The OS installed on the machines is Ubuntu 11.10, 32bit running no graphical interface. The algorithms were implemented using C++. On each machine only one experiment was run at a time to minimize side-effects. Furthermore, all programs used only one single core.

7.1 Environments

The experiments will be run on some of Saaltink's original environments, as well as some newly created environments. The reasons for this are that most of the environments used by Saaltink are very simple, have a small number of vertices or exist mainly out of one single Topo-Forest (almost all vertices have an overlap with some other vertex). To allow all methods to be tested, additional environments have to be created. The environments will be described in Section 7.1.2 after the different types of environments have been introduced.

7.1.1 Environment types

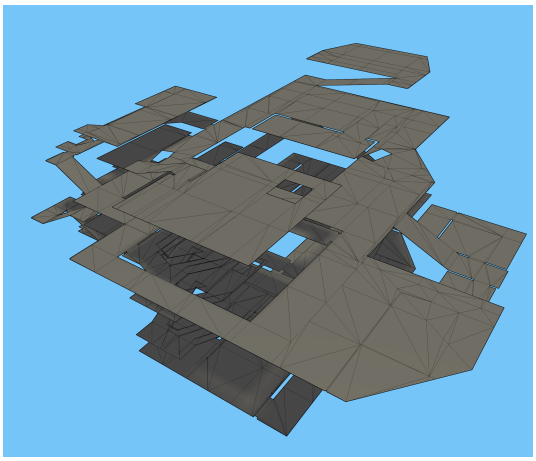
Environments can basically be divided into separated and unseparated environments. A separated environment is an environment consisting out of multiple TFs, whereas an unseparated environment exists out of only one single TF. This distinction is important since some of the reduction algorithms described in Chapter 4 need vertices without overlap. Others need a clear separation of different structures in the environment to work (the TF method).

Another distinction that can be made is the number of floors/layers in the environment. The number of overlaps can increase faster than the number of edges in a layered environment. In a worst case scenario, the number of overlaps can be $O(\sum_{i=1}^{|V|-1} i) = O((|V|-1)^2 + |V|)$ while the number of edges always increases in a linear fashion with respect to the number of vertices. Since the number of edges increase linearly, this means that environments with a lot of layers have a completely different ratio of overlaps/edges. To test the effectiveness of the algorithms on all possible environments, it is important that also this distinction is taken in consideration.

7.1.2 Environment descriptions

The environments used are described below. The 'Type' of an environment can either be separated or unseparated. For separated environments, 'TF' is set to the number of TFs present in this environment. This number was found using the algorithm described on page 24. '$|V|$', '$|E|$' and '$|O|$' refer to the number of vertices, edges and overlaps in the environment. '$\frac{|O|}{|V|}$' gives the ratio between overlaps and edges, a way to measure how layered a certain environment is. Whenever there is a known minimum for the number of cuts needed it will be listed under 'min $|T|$'. Whenever the environment is a model of an existing environment, it will say so under 'Existing'. Finally, as short description of the environment will be given.

as_oilrig

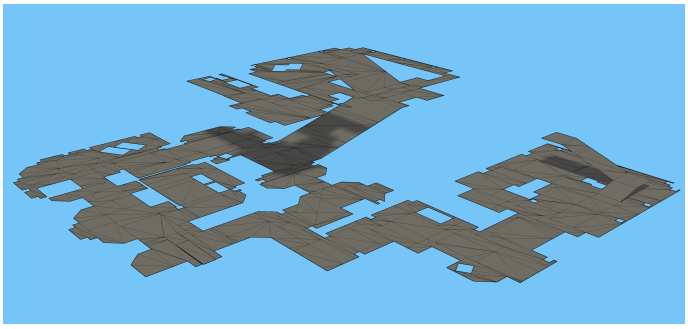


Type: Separated
TFs: 2
|V|: 634
|E|: 959
|O|: 668
$\frac{|O|}{|V|}$: 1.05
min |T|: Unknown
Existing: No
Short description:
This environment is one of Saaltink's original environments. It was taken from a game level. As the name suggests, it is the floor plan of an oilrig.
The reason for using this environment is the simplicity of the environment, as well as the fact that this environment is an actual game level. Because it is a game level, it is that this environment shows some typical structures for this type of environments.

as_oilrig_scaled

Type: Separated
TFs: 5
|V|: 1830
|E|: 2150
|O|: 5934
$\frac{|O|}{|V|}$: 3.24
min |T|: Unknown
Existing: No
Short description:
This environment is a scaled version of *as_oilrig* and is reused in the environments *tf1*, *tf2* and *tf3*. When importing this file in Blender to use as a part of the aforementioned environments, quite some of the polygons got triangulated. The resulting version of *as_oilrig* turned out to be a more complex environment in terms of vertices, edges, overlaps and the number of TFs.

de_vertigo

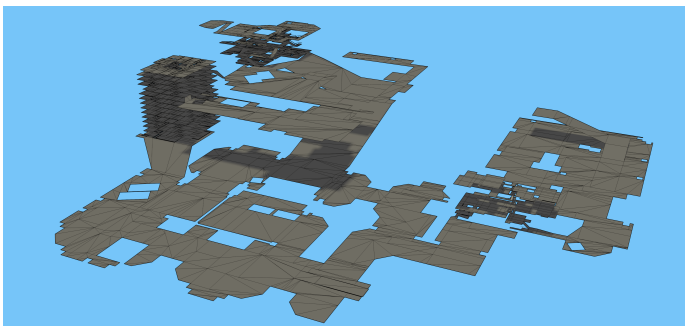


Type: Separated
TFs: 5
|V|: 367
|E|: 642
|O|: 37
$\frac{|O|}{|V|}$: 0.10
min |T|: 2
Existing: No
Short description:
This environment is another one of Saaltink's original environments. It was taken from a game level of the computer game Counter Strike Source. This environment shows a relatively simple environment with very little overlapping vertices.

de_vertigo_scaled

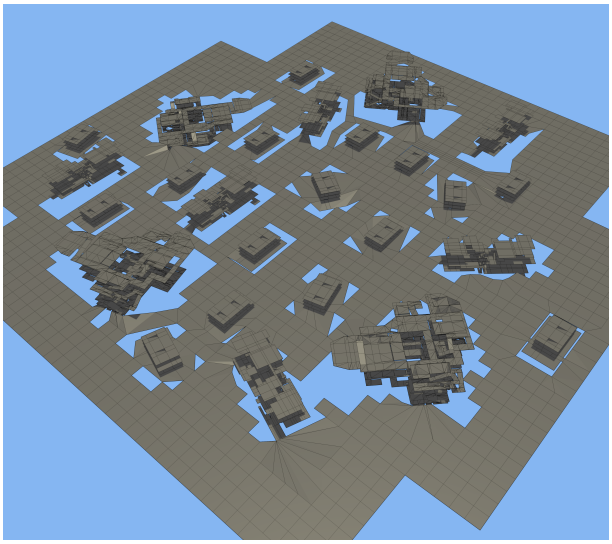
Type: Separated
TFs: 6
|V|: 1196
|E|: 1480
|O|: 409
$\frac{|O|}{|V|}$: 0.34
min |T|: Unknown
Existing: No
Short description:
This environment is the scaled version of *de_vertigo*. Just as *as_oilrig_scaled*, it was partially triangulated when imported in Blender. This environment is also used as a part of the environment *tf1*.

tf1



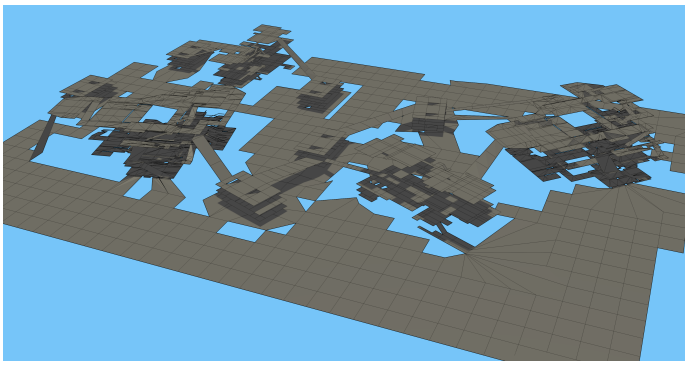
Type: Separated
TFs: 14
|V|: 9190
|E|: 12000
|O|: 130536
$\frac{|O|}{|V|}$: 14.2
min |T|: Unknown
Existing: No
Short description:
This environment is a combination of *as_oilrig*, *de_vertigo*, *max* and *uulib*. A big part of its vertices, overlaps and edges are located inside the *max* environment. *tf1* has several TFs that can be used as subproblems, as well as some TFs that cannot be used as subproblems.

tf2



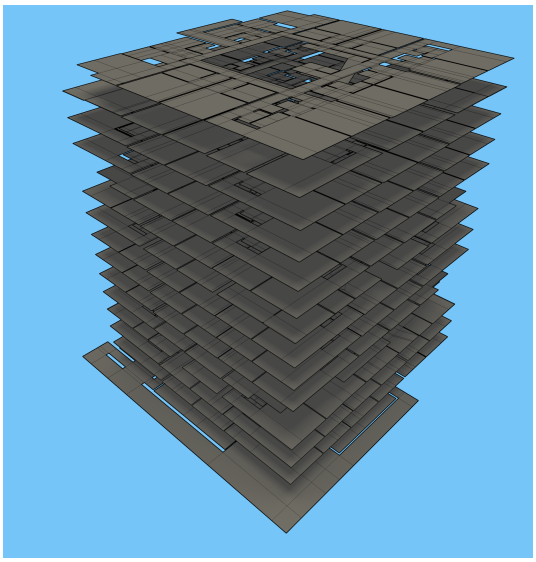
Type: Separated
TFs: 41
|V|: 11574
|E|: 15300
|O|: 38562
$\frac{|O|}{|V|}$: 3.33
min |T|: Unknown
Existing: No
Short description:
This environment is a combination of 4 *as_oilrig*, 6 *uulib* and 16 small garage-like buildings. The large number of smaller buildings makes it look like a virtual version of an urban environment. The environment was constructed in such a way that almost all TFs should deliver reusable results when solving the `MIN-T-MLE` for each individual TF. The fact that a lot of the TFs are reusable will hopefully result in a speed-up when finding a `MIN-T-MLE`.

tf3



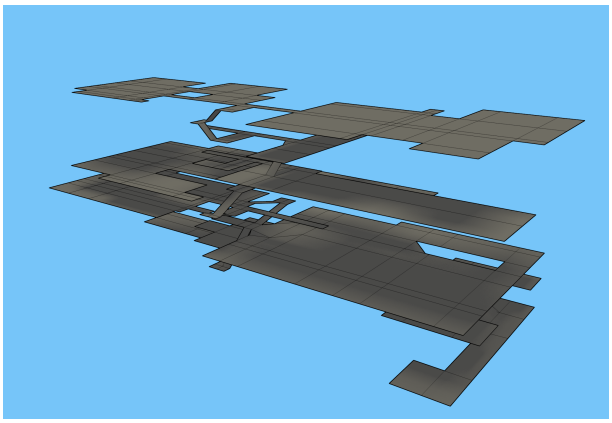
Type: Separated
TFs: 16
|V|: 5179
|E|: 6871
|O|: 18387
$\frac{|O|}{|V|}$: 3.55
min |T|: Unknown
Existing: No
Short description:
This environment is a combination of 2 *as_oilrig_scaled*, 2 *uulib* and 6 garage-like buildings. Just as *tf2* it was modelled to look like an urban environment. Instead of incorporating the individual structures in the environment in such a way that using TFs should result in a speed-up, the structures are all interconnected in such a way that it should result in a slowdown.

max



Type: Unseparated
TFs: 1
|V|: 5933
|E|: 8034
|O|: 112945
$\frac{|O|}{|V|}$: 19.03
min |T|: 28
Existing: Based on
Short description:
This environment is based on one of the towers of City Campus Max in Utrecht. It has 15 stories, each floor connected to the other by double stairs. On each floor several rooms exists, often attached to the remainder of the building by one single polygon. This makes it particularly interesting to test the overlap- and edge-removal methods. Furthermore, this environment has by far the highest $\frac{|O|}{|V|}$ ratio. Some of the algorithms, like Saaltink's heuristic (Section 5.1) as well as the genetic algorithm (Section 5.3) are greatly influenced by this ratio.

uulib



Type: Unseparated
TFs: 1
|V|: 298
|E|: 421
|O|: 813
$\frac{|O|}{|V|}$: 2.72
min |T|: 8
Existing: Yes
Short description:
This environment is based on the University Library in Utrecht. This building is interesting because of the fact that it exists in real-life. Therefore, it is assumed that it shows some typical real-life structures. Furthermore, it is one of the few environments of which the size of the minimal set of transfers is known.

7.2 Preprocessing

Every environment can be pre-processed using the methods described in Section 4.3. During these experiments, all described environments are processed using Saaltink's methods (`1-CONTRACT` and `2-CONTRACT`), the new edge reduction step (`E-REDUCE`) and the new overlap

removal methods (d-REMOVE and STACK-REMOVE).

In a first experiment, all environments are pre-processed using both 1-CONTRACT and 2-CONTRACT. During the second experiment, E-REDUCE will be added to the set of operations to check out the effect it has on the number of possible applications of 1-CONTRACT and 2-CONTRACT.

In a third experiment the overlap-removal methods d-REMOVE and STACK-REMOVE will be used to reduce the number of overlaps in the environment. Parameter d will be varied from 1 to 3 during these experiments. The fourth and last experiment will combine Saaltink's methods with both the overlap-removal methods and the E-REDUCE operation in an attempt to make the environments as compact as possible. Once again d will be varied between 1 and 3 during these experiments.

During all experiments, the total runtime and the number of applications of each individual operation will be counted, as well as the number of vertices, edges and overlaps removed. Each individual experiment will be repeated 20 times. The experiment is repeated to account for differences in execution times in the case of Saaltink's reductions 1-CONTRACT and 2-CONTRACT as well as to account for the random order in which overlaps are removed using d-REMOVE.

7.3 Heuristic methods

All the heuristic methods will be run on both the original as the most compact versions of the environments obtained during the experiments done with the graph reductions.

7.3.1 Local search

For all variants of local search, the parameters that can be changed are p_E, p_S and p_O. These parameters have values in the interval $[0, 1]$ and $p_E + p_S + p_O = 1$. p_E encodes the chance that MERGE will be executed, p_S the chance for SPLIT and p_O the chance of MOVE. Furthermore, our implementation of local search also supports changing the likelihood an operation is picked over time. This can be useful since the only non-destructive operation implemented is the MERGE operator. Increasing the probability of picking the MERGE operation might increase the speed at which good solutions are found, but eventually the destructive operations SPLIT and MOVE are needed to move away from a local optimum. The chance an operation is picked can be manipulated in one of two ways:

1. Through an instantaneous increase or decrease whenever no better solution can be found, or;

2. Through a continuous increase or decrease over time towards a predetermined value at a predetermined speed.

The last parameter of local search is a simple termination parameter t, forcing the search to terminate after t unsuccessful attempts at improving the current best score.

For simulated annealing there exist two more parameters. The starting temperature and the decrease in temperature will be tweaked such that for the environments where a minimal cut is known, this value can be found. For all simulated annealing experiments, the MERGE operation is combined with SPLIT and MOVE. For simulated annealing, six different

configurations were tested. Three of the configurations used a high starting temperature of 100. Every 1000 iterations the current temperature was multiplied by 0.99. The other three configurations had a low starting temperature of 10 . This temperature was multiplied by 0.9 every 500 iterations. The reason for the big difference between these two configuration is that these configurations are also used in combination with the dynamic chance-update. Using the high starting temperature we tried to prevent that simulated annealing would have a low temperature by the time the dynamic chance-update finished modifying the chances that the different operations were picked.

For each temperature setting, a configuration with no dynamic chance-update was used. During these experiments the chances an operation would be picked were all equal. Another configuration was run which started out using only the `MERGE` operator, switching to equal chances as soon as no better solution was found. The last two configurations also started only using the `MERGE` operation. During each individual iteration the chances were updated so that at iteration 10000 all chances were equal again.

For tabu-search, the same experimental set-up is used, with two additional parameters, the neighbourhood-size and the size of the tabu list. Instead of a chance for a certain operation being picked, each operation will be responsible for generating a part of the neighbour-list proportional to the chance it will be picked. Two distinct configurations of tabu-search were used during these experiments. The first configuration had a neighbourhood-size of 5 and the tabu list had a size of 10000. During a second experiment a neighbourhood-size of 10 was picked and the tabu list had a size of 1000. The chances for picking the different operations were all equal. An explenation of what is stored in the tabu list and what the neighbourhood-size paramter exactly does is described in Section 5.2.3.

Every experiment was stopped after no better solution was found for 10000 iterations or at most 1800 seconds passed. All experiments were repeated 10 times.

During all experiments the total runtime as well as the time needed to find the best solution will be tracked. Furthermore, the current score of the active solution will be tracked. Whenever a operation is picked the found change in score will be remembered in combination with at which iteration this happened, as well as if the change was accepted or not.

7.3.2 Genetic algorithm

The genetic algorithm as implemented has five different parameters. The first one is the mutation rate. The second parameter is the size of the grid for the recombination operator. The parameter for the recombination operator b is an integer depicting how much vertical and horizontal bins should be created. This means that if b is set to a value of x, the number of bins created equals x^2. Furthermore, it is possible to change how often the recombination operator is applied by manipulating the parameter r. When $r = 1$, the recombination operator is used exclusively. Parameter t is a simple termination parameter, forcing the genetic algorithm to terminate after t unsuccessful attempts at improving the current population. The last parameter s is the maximal number of seconds the algorithm is allowed to search for a solution.

Experiments will be done using only the mutation operation as well as with both operations. During the experiments the size of the population, the average fitness scores of the solution and the total runtime will be monitored.

During the first experiment, an average of one change will occur for each mutation. This

is the same as Neumann et al [34] had in their experiments. In a second experiment the mutation rate will be increased. In the third and fourth experiment, the re-combinator will be introduced. During both experiments b will be set to 40. In the third experiment r will be set to 0.3 and in the fourth experiment r will be set to 0.5. When no better solution has been found in 1000 iterations or 1800 seconds, the search is stopped.

The goal of these experiments is to determine if the recombination operator is useful or not. It will not be compared with any of the other algorithms because it has not been optimized in any way. Therefore, we feel like comparing it to any another algorithm would be unfair.

7.3.3 Shortest Path Heuristic

This heuristic does not have any parameters that can be tweaked. During the experiments the total runtime as well as the number of shortest path calculations will be counted. Since this method was designed to quickly attempt to find the stairs/ramps in real buildings, we will also track the actual cuts performed by this algorithm, to check if the algorithm is able to find them. Furthermore, the total runtime will be measured. The experiment will be repeated 20 times.

7.3.4 Height Heuristic

This heuristic does not have any parameters that can be tweaked. During the experiments the total runtime as well as the number of cuts for the solution after clustering, merging and redistributing the vertices will be tracked. The experiment will be repeated 20 times.

7.4 Column generation

For column generation, the pricing problem can be solved using the ILP (Section 6.2.1), a local search method (Section 6.2.2) or a two-phase method where in the first phase of a pricing the local search method is used. Whenever the local search does not find a better solution, the ILP is used to determine if there really does not exist a better solution. In Section 7.4.1 the best parameters for the local search will be estimated. Section 7.4.2 will describe the experiments in which the actual problem will be solved.

7.4.1 LS as pricer

The goal of the experiments described here will be to find the best parameters for a local search meta-heuristic for solving the pricing problem as close to optimality while remaining fast. For this purpose, all vertices of the graphs will be weighted to simulate the effect of the shadow prices of the vertices, π_v. The random values will be picked in the ranges $[0, 1]$, $[0, 2]$ and $[0, 3]$. Using the ILP Pricer, we will search for the optimal value for the reduced cost for this particular graph instance. This value will be stored and used to compare the performance of the different local search instances. During the experiments, the chance that either `ADD`, `REMOVE` or `SWITCH` will be picked, will all be set to $\frac{1}{3}$. Preliminary experiments have shown that tabu-search outperforms simulated annealing. For that reason only one configuration

will be using simulated annealing. For this configuration the stating temperature was set to 50. This temperature was multiplied by 0.99 every 100 iterations.

Two configurations using tabu-search were tested. One of the configurations had a neighbourhood-size of 10, the other configuration used a neighbourhood-size of 5. For both configuration the maximum size of the tabulist was 1000.

During these experiments no dynamic change of the chances for picking each operation will be used. The parameters being tracked can be found in Section 7.3.1. The experiments were repeated 50 times.

7.4.2 LS as pricer combined with ILP as pricer

With the help of Section 7.4.1, a configuration of local search is determined. Using this local search pricer and the ILP pricer, an attempt is made to find the optimal solution for each individual environment within eight hours using Branch-and-Price. When no optimal solution is found within allocated time, the original ILP is solved for the current set of columns.

During all the experiments the number of columns generated as well as the number of columns generated by each pricer, the solutions found and the total run-time will be tracked. If no optimal solution was found, the optimal solution for the restricted set of columns as well as the time needed to solve for these columns will be reported.

7.5 Topo-Forests

For testing the divide and conquer method, the best tested versions found for column generation, local search and the genetic algorithm will be used to determine the effect on the time needed for finding an optimal solution.

Using the best local search, Topo-Forests will be partitioned and reduced whenever this is possible. Next, the problem for the reduced graph will be solved using the best performing version of local search. During each experiment, the time needed until termination will be tracked, as well as the total number of cuts found.

Chapter 8

Discussion of results

In this chapter the results found during the experiments will be discussed. All the results are put in appendices A through I. Results of different configurations are deemed statistically different at or below the 5%-significance level. These values are obtained using either Anova using Tukey as post-hoc analysis or using a simple t-test.

Each individual section in this chapter treats one set of experiments. When examples are given in these sections, these examples are for the environment as_oilrig unless another environment is explicitly mentioned.

8.1 Reduction experiments

The results of all the experiments are given in Appendix A. For each individual environment the absolute reductions (e.g. Table A.1) and the relative reductions (e.g. Table A.2) of $|V|$, $|E|$ and $|O|$ are given. Furthermore, tests were done to determine the statistical relevance of the addition of the `E-REDUCE` operation (e.g. Table A.3), the impact of changing d for `d-REMOVE` (e.g. Table A.4) and the application of all operations (e.g. Table A.5). Table A.46 gives the details of the environments used for further experimentation.

8.1.1 `E-REDUCE`

The addition of `E-REDUCE` is shown to be significant for all the environments except max and uulib. For these environments the number of vertices are not reduced significantly when comparing the results obtained by using only `SAAL` or `SAAL` and `E-REDUCE` combined.

For all other environments the number of both vertices and edges in the resulting environment are reduced. The increase in reduction in the number of vertices compared to only using Saaltink's operations, ranges from 2 for as_oilrig_scaled to 26 for de_vertigo_scaled. The reduction in the number of edges ranges from 3 as_oilrig to 44 for de_vertigo.

Although the number of vertices and the number of edges are only reduced very little compared to Saaltink's methods, it is still deemed statistically significant since each individual run resulted in exactly the same score.

8.1.2 `d-REMOVE`

For all environments, the number of overlaps is significantly reduced. The number of overlaps are reduced within a range of 25.5% for de_vertigo with $d = 1$ to 67.1% for tf2 with $d = 3$. In

all cases except for de_vertigo increasing d to two results in another significant reduction in the number of overlaps.

Unfortunately, the application of these operations can take up a considerable amount of time. For example, for the environment tf1 it took an average of 1190.46 seconds to reduce the number of overlaps whenever $d = 3$.

8.1.3 `d-REMOVE` combined with `E-REDUCE`

Combining all different operations resulted in a further significant decrease in the number of overlaps, vertices and edges. For the environments as_oilrig, de_vertigo and de_vertigo_scaled, the results for $d = 3$ compared to $d = 2$ is not significant for $|V|$ and $|E|$. For the environment de_vertigo even $|O|$ is not significantly reduced when comparing $d = 3$ to $d = 2$. A reason for this might be that $|O|$ was fairly small to begin with. Therefore, it can be the case that the minimal size of $|O|$ might already be obtained.

8.2 SPH experiments

The results for the Shortest Path heuristic are described in Appendix B in Table B.1. As can be seen, most of the time was spent cutting the shortest path and finding new shortest path after the initial run of Johnson's algorithm has completed. Another thing to take note of is that for all environments the minimized versions have a shorter runtime. The difference between the original and the minimized environments will be discussed in Section 8.9.

8.3 HH experiments

The results for the Height Heuristic are given in Appendix C. Table C.1 shows the results for the `HH` algorithm after each of the three phases. Table C.2 shows the change in score between the different phases. The first thing to note is the quality of the solutions listed in the column $|T|R$ and the little amount of time needed. For all environments the algorithm completes within a few seconds. Another thing to note is that for five out of nine environments, the solution changes significantly after each different phase. For the environments max and max_min, no significant change is made after the initial cluster and component creation phase. In this first phase, the optimal solution for that specific environment has already been found. This is not very strange since this particular environment has each individual floor at the same height level and the transfers between the different layers at different heights.

A possible explanation for the good results for `HH` is that the environments consist out very little sloped surfaces. The reason for this is that this heuristic has been added at the very end of the experiments. Therefore, no worst-case environments were constructed. However this type of environment can exist, espacially in game levels.

8.4 GA experiments

The results for the experiments done using a genetic algorithm are given in Appendix D. Because of the small sample sizes, nothing significant can be said about the quality of the

solutions when comparing the different results of only the valid solutions found during these experiments. When comparing the number of cuts found for any solution, still no statistical significance is found.

However, when comparing the number of valid solutions found when using the default settings or default settings in combination with a recombination operator there seems to be a small difference favouring the recombination operator. Besides that more feasible solutions were found, the quality of these feasible solutions is usually higher than that of the results obtained for experiments done without the recombination operator. However, to determine if this is statistically significant further tests should be conducted.

8.5 LS experiments

The results of all the experiments done during these experiments are given in Appendix E. For each individual experiment three tables are listed. The first table lists the cut-size found, as well as the time until best (TTB), iterations until best (ITB), total runtime (TTE) and iterations until end (ITE). The second and third table describe if there were found any statistical differences between the different configurations. The second table shows at what significance level there was a statistical difference between the different configurations when comparing the quality of the solution. The differences between the TTE values for the different configurations are compared in the third table.

The tabu-search configuration with neighbourhood-size 5 is listed as Tabu 5, whereas the version with neighbourhood-size 10 is listed as Tabu 10. All simulated annealing configurations are called 'Sim.'. Tmin is added whenever it is one of the three configurations with starting temperature 10. An L is added for configuration that have constantly changing chances of picking an operation. An I is added for the configuration that change instantaneously.

8.5.1 Comparing scores

The first thing that catches the eye is that the 'Sim.' configuration has relatively bad scores for all environments with the exception of uulib_min, de_vertigo_scaled_min, de_vertigo_scaled and de_vertigo. The reason for this is probably that the value of T was too high. 'Sim. I' and 'Sim. L' were meant to counteract this by forcing a lot of merges in the beginning, effectively turning simulated annealing into a hillclimber. As can be seen both configurations perform almost always better than 'Sim.' on the bigger environments such as max, tf1, tf2 and tf3. On both de_vertigo_scaled and de_vertigo_scaled_min the performance is less than that of 'Sim.'. This might be due to the 'Sim. L' and 'Sim. I' configuration getting stuck in local minima.

Both 'Tabu' configuration perform very good when only considering the values of the solutions found. However, these configurations are very slow.

8.6 Saaltink's experiments

The results for the algorithms developed by Saaltink are described in Appendix F in Table F.1. As can be seen, the flood-fill algorithm is very fast, but yield decompositions of environments that are not very good. The flow-based approach (listed under cut) yields better results, at

a cost of rapidly increasing runtime. For some of the environments the flow-based approach was not tested because the algorithm was too slow.

8.7 ILP experiments

8.7.1 Heuristic pricer

The results for the experiments for the pricer are given in tables G.1 through G.4. Tabu 1 is the configuration of tabu search with a neighbourhood size of 10. As can be seen in Table G.2 and G.4, there is a significant difference between every configuration. Configuration Tabu 1 performs significantly better than any other configuration, but lasts significantly longer. Still the configuration Tabu 1 is our configuration of choice. The reason for this is that the ILP Pricer can run for several hours. Picking a LS Pricer that finds better solutions probably also means picking a LS Pricer that is more likely to find columns of positive reduced cost, avoiding the execution of the ILP Pricer.

8.7.2 Branch-and-Price

In tables G.5 and G.6 the results for the ILP experiments are given. When only using the LS Pricer (Table G.5), known minima are found for the environments uulib, and de_vertigo. For the environment de_vertigo_scaled only a minimal decomposition is found for the original environment. Only for the non-scaled versions of de_vertigo, the Branch-and-Price method also proved that it found the minimal decomposition.

The version of Branch-and-Price using the Height Heuristic shows very different results. For certain environments for which the minimum was found using only local search, this version is not able to find these minima. For some environments, a big improvement is found (e.g. for tf1 an improvement of 542 was found). This improvement was expected when looking at Section 8.3.

8.8 TF

The results for the experiments done using TFs are given in Table H.1. When comparing the found solutions for the different solutions the value for each environment is worse than found using plain local search. A possible reason for this might be that the TF structure is not as suitable for selecting entire buildings as intended. Only the environments max and uulib consist out of a single TF. The environments as_oilrig and as_oilrig_scaled exist out of multiple TFs while it is one single building. This shows that a TF as described in Chapter 4 is not sufficient for selecting complete buildings.

8.9 Differences between original and minimized environments

In Appendix I an analyses of the influence of minimized environments for `SPH`, `HH`, `LS` and `TF` are given. The first thing that stands out is that almost all runs had a significant difference in execution time in favour of the minimized environments.

When looking at the solution quality, all environments for SPH that showed a significant difference were also in favour of the minimized environments, with the exception of the environments de_vertigo_scaled. For HH the opposite is true, whenever there was a significant difference in score, it was in favour of the original environments (with the exception of tf3). This can be explained by what happens when environments are reduced. In this process, polygons at different height levels can end up in the same vertex of the **P**-Graph. This again can disturb HH since artificial slopes are introduced, as well as new height levels (located between the different height levels of the original polygons). However, since reducing an environment usually takes more time than it takes to run HH, this is not such a big problem.

For local search the same observation can be made. In most cases, when there is a significant difference in score, it is in favour of the original environments. Notable exceptions are the environments as_oilrig_scaled, tf1 and tf2. Why these environments do have statistically significant improved scores when mimized, and the other environments do not, is not known. The last analysed algorithm, TF also shows worse solution values for the environments if they are reduced.

Chapter 9

Conclusion

Since it is possible to reduce P3Sr to MIN-T-MLE and there are no bounds on the treewidth for **P**-Graphs, the only chance of finding good solutions for MIN-T-MLE is using heuristic methods. Using Branch-and-Bound or Branch-and-Price simply takes too much time. Several methods have been formulated and benchmarked in this thesis. In the beginning, an attempt was made to reduce the problem instances. How useful this is, will be told in Section 9.1. A conclusion about what method to use will be given in Section 9.2. In Section 9.3 a short list of unanswered problems will be given.

9.1 Reducing environments

For reducing environments, new methods have been formulated that are able to substantially reduce the size of environments. Unfortunately, the quality of the solutions for a lot of the algorithms decreases. All but SPH showed a decrease in the solution quality. Furthermore, the speed-up gained by first reducing an environment is not big enough to warrant this additional step. For example, the speed-up for the environment max when looking at local search is less than one minute, while reducing the environment (when using all different operations) took between five and fifteen minutes. Saaltink already concluded in his thesis [40] that only using his reductions, which are the most basic reductions possible, is not profitable.

9.2 Which method to use

All experiments were benchmarked on nine environments and their reductions. All methods except the ILP using Branch-and-Price offer no guarantee of optimality. The downside of using the ILP in combination with Branch-and-Price is the running time. All experiments done using this method were cut off after eight hours, but in most cases no optimum was found within the time limit.

HH yields very good results on the tested environments in a very short time. However, this method was only added at the last moment. Because of this no environments were generated for which HH is likely to be outperformed by other methods. These environments would consist out of sloped environments.

As mentioned in the discussion, the results for TF were not as good as expected. One possible reason is that the TF-structure is not sufficient for selecting entire 'buildings'. This idea is supported when looking at the environments used. For example the environment

as_oilrig consists out of one 'building', but is comprised out of multiple TFs. SPH proved very sensitive to the number of overlaps, which was to be expected. Furthermore, the quality of the solutions were worse than those of other methods like LS.

LS as implemented had several different configurations, of which eight configurations were tested. Although the tabu search configurations obtained very good results for most environments, the time needed a multitude of some of the simulated annealing configurations. Of the simulated annealing configurations, there are two configuration that stick out. The first one is plain simulated annealing (Sim.). This version usually found bad results, probably because of a lack of incentive to improve. This is due to the slow decrease in temperature. The other configuration is simulated annealing with reduced temperature (Sim. Tmin). This configurations usually took significantly more time to terminate. Between the remaining four configurations there is no statistically significant difference.

The results for each individual environment and algorithm are plotted in a box-plot in Appendix J. These plots make clear that the flood-fill partitioner, flow partitioner, SPH and TF methods are outperformed in most situations. It also emphasizes the good performance of HH. That, together with the great speed suggests that HH should be the method of choice for environments with very little sloped polygons. Most of the environments we tested on were of this type. For environments with a lot of slopes, no tests have been made, however we expect it to perform not as good for these types of environments.

For all other environments a type of local search is recommended. The use of tabu searches as well as 'Sim.' and 'Sim. Tmin' are discouraged because of previously mentioned reasons. Between the remaining four versions of local search, no significant differences can be found. Therefore, the use of one of these types of local search is advised for environments with sloped surfaces.

9.3 Open problems

There are still a number of open problems. The first one concerns the hardness of MIN-T-MLE. This problem looks very similar to the MULTICUT problem, for which is known that it is also APX-Hard. The question is if this also holds for MIN-T-MLE.

Another open problem that remains concerns the graph reductions. Using the current implementation, overlaps were removed from the **P**-Graphs as soon as it was proven that that particular overlap was already guaranteed. Because of this, the order in which overlaps were removed, mattered for how much overlaps could be removed. Finding an optimal order is one open problem.

Since most of this thesis was based on a conversion of a polygonal environment to a graph on which the problem is NP-Hard, another open problem is if there is another way to formulate the problem (maybe avoiding graphs all together). It might be the case that using this different formulation, employing more information (for example more of the height information of the individual polygons), might make it possible to find better, faster heuristics.

Bibliography

[1] *The boost graph library: user guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[2] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[3] R.K. Ahuja and J.B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

[4] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[5] C. Bentz. A simple algorithm for multicuts in planar graphs with outer terminals. *Discrete Applied Mathematics*, 157(8):1959–1964, 2009.

[6] H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[7] J.M. Boyer and W.J. Myrvold. On the cutting edge: Simplified $o(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8:2004, 2004.

[8] M. Brinkmeier. A simple and fast min-cut algorithm. *Theory of Computing Systems*, 41:369–380, 2007. 10.1007/s00224-007-2010-2.

[9] G. Calinescu, C.G. Fernandes, and B. Reed. Multicuts in unweighted graphs with bounded degree and bounded tree-width. In *Integer Programming and Combinatorial Optimization*, pages 137–152. Springer, 1998.

[10] V. Ĉerný. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985. 10.1007/BF00940812.

[11] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 144–153, 2005.

[12] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[13] M. Cyrus and J. Beck. Generalized two- and three-dimensional clipping. *Computers and Graphics*, 3(1):23 – 28, 1978.

[14] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[15] P. Elias, A. Feinstein, and C. Shannon. A note on the maximum flow through a network. *Information Theory, IEEE Transactions on*, 2(4):117–119, 1956.

[16] L.R. Ford and D.R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404.

[17] L.R. Ford and D.R. Fulkerson. Solving the transportation problem. *Management Science*, 3(1):24–32, 1956.

[18] A. Fournier and D.Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3(2):153–174, 1984.

[19] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[20] N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997.

[21] R. Geraerts. Planning short paths with clearance using explicit corridors. In *IEEE International conference on robotics and automation*, pages 1997–2004, 2010.

[22] F. Glover and C. McMillan. The general employee scheduling problem. An integration of MS and AI. *Computers & Operations Research*, 13(5):563–573, 1986. Applications of Integer Programming.

[23] J. Guo, F. Hüffner, E. Kenar, R. Niedermeier, and J. Uhlmann. Complexity and exact algorithms for multicut. In *Software Seminar*, pages 303–312, 2006.

[24] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48:723–760, July 2001.

[25] J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, October 1974.

[26] T.C. Hu. Multi-Commodity Network Flows. In *Operations Research*, volume 11, pages 344–360, 1963.

[27] A. Itai. Two-commodity flow. *J. ACM*, 25:596–611, 1978.

[28] D.B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[29] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[30] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[31] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[32] S.M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[33] D. Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[34] F. Neumann and J. Reichel. Approximating minimum multicuts by evolutionary multi-objective algorithms. In *Parallel Problem Solving from Nature PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 72–81. Springer Berlin / Heidelberg, 2008.

[35] F. Neumann, J. Reichel, and M. Skutella. Computing minimum cuts by randomized search heuristics. Technical report, Collaborative Research Center 531, Technical University of Dortmund, 2008.

[36] N. Robertson and P.D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39 – 61, 1983.

[37] N. Robertson and P.D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

[38] N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153 – 190, 1991.

[39] S.J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach (International Edition)*. Pearson US Imports & PHIPEs, November 2002.

[40] W. Saaltink. Partitioning polygonal environments into multi-layered environments. Master's thesis, Utrecht University, 2011.

[41] A. Schrijver. *Combinatorial Optimization - Polyhedra And Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[42] W. van Toll. A navigation mesh for efficient density-based crowd simulation in multi-layered environments. Master's thesis, Utrecht University, 2011.

[43] W. van Toll, A.F. Cook IV, and R. Geraerts. Navigation meshes for realistic multi-layered environments. In *Intelligent Robots and Systems*, pages 3526–3532. IEEE, 2011.

[44] E.W. Weisstein. Möbius Strip. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/MoebiusStrip.html`.

[45] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. `http://www.zib.de/Publications/abstracts/TR-96-09/`.

Appendix A

Reduction results

A.1 as_oilrig

	|V|	(dev)	|E|	(dev)	|O|	(dev)	t (s)	(dev)
Original	634.	---	959.	---	668.	---	---	---
Saaltink	525.	0.	849.	0.	---	---	0.	0.
Saal.+E-REDUCE	521.	0.	827.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	479.55	2.24	0.12	0.02
2-REMOVE	---	---	---	---	459.95	1.73	0.22	0.03
3-REMOVE	---	---	---	---	453.85	1.14	0.49	0.06
Saal.+E-R.+1-R.	502.1	0.91	805.3	1.17	473.2	2.93	0.19	0.01
Saal.+E-R.+2-R.	498.65	0.81	801.8	1.11	456.75	1.8	0.31	0.01
Saal.+E-R.+3-R.	498.7	1.81	801.4	2.26	450.2	2.59	0.68	0.01

Table A.1: The absolute results of applying different reduction operations on the environment as_oilrig. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.828	0.	0.885	0.	---	---
Saal.+E-REDUCE	0.822	0.	0.862	0.	---	---
1-REMOVE	---	---	---	---	0.718	0.003
2-REMOVE	---	---	---	---	0.689	0.003
3-REMOVE	---	---	---	---	0.679	0.002
Saal.+E-R.+1-R.	0.792	0.001	0.84	0.001	0.708	0.004
Saal.+E-R.+2-R.	0.787	0.001	0.836	0.001	0.684	0.003
Saal.+E-R.+3-R.	0.787	0.003	0.836	0.002	0.674	0.004

Table A.2: The relative results of applying different reduction operations on the environment as_oilrig. The columns $|\mathbf{V}|$, $|\mathbf{E}|$ and $|\mathbf{O}|$ list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.3: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing $|\mathbf{V}|$ and Table (b) gives the found significance levels when comparing $|\mathbf{E}|$.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.4: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying d-REMOVE for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.5: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+E-REDUCE+d-REMOVE for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.2 as_oilrig_scaled

	|V|	**(dev)**	**|E|**	**(dev)**	**|O|**	**(dev)**	**t (s)**	**(dev)**
Original	1830.	---	2150.	---	5934.	---	---	---
Saaltink	1513.	0.	1832.	0.	---	---	0.	0.
Saal.+E-REDUCE	1511.	0.	1829.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	2480.75	20.72	2.85	0.25
2-REMOVE	---	---	---	---	2153.15	17.64	4.26	0.23
3-REMOVE	---	---	---	---	2001.75	13.07	5.84	0.44
Saal.+E-R.+1-R.	1295.8	6.48	1612.8	6.48	2325.55	17.44	3.54	0.21
Saal.+E-R.+2-R.	1260.65	3.63	1577.65	3.63	2035.6	12.58	4.17	0.04
Saal.+E-R.+3-R.	1249.4	3.56	1566.4	3.56	1904.4	12.73	5.99	0.22

Table A.6: The absolute results of applying different reduction operations on the environment as_oilrig_scaled. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.827	0.	0.852	0.	---	---
Saal.+E-REDUCE	0.826	0.	0.851	0.	---	---
1-REMOVE	---	---	---	---	0.418	0.003
2-REMOVE	---	---	---	---	0.363	0.003
3-REMOVE	---	---	---	---	0.337	0.002
Saal.+E-R.+1-R.	0.708	0.004	0.75	0.003	0.392	0.003
Saal.+E-R.+2-R.	0.689	0.002	0.734	0.002	0.343	0.002
Saal.+E-R.+3-R.	0.683	0.002	0.729	0.002	0.321	0.002

Table A.7: The relative results of applying different reduction operations on the environment as_oilrig_scaled. The columns |**V**|, |**E**| and |**O**| list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.8: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing |**V**| and Table (b) gives the found significance levels when comparing |**E**|.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.9: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying d-REMOVE for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of |**O**|.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.10: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+E-REDUCE+d-REMOVE for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of |**V**| (Table (a)), |**E**| (Table (a)) and |**O**| (Table (a)).

A.3 de_vertigo

	|V|	**(dev)**	**|E|**	**(dev)**	**|O|**	**(dev)**	**t (s)**	**(dev)**
Original	367.	---	642.	---	37.	---	---	---
Saaltink	269.	0.	543.	0.	---	---	0.	0.
Saal.+E-REDUCE	264.	0.	499.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	27.55	0.51	0.	0.01
2-REMOVE	---	---	---	---	26.6	0.5	0.01	0.
3-REMOVE	---	---	---	---	26.5	0.51	0.03	0.
Saal.+E-R.+1-R.	256.	0.	487.	0.	27.2	0.41	0.01	0.01
Saal.+E-R.+2-R.	255.45	0.51	485.9	1.02	26.45	0.51	0.01	0.
Saal.+E-R.+3-R.	255.4	0.5	485.8	1.01	26.7	0.92	0.03	0.01

Table A.11: The absolute results of applying different reduction operations on the environment de_vertigo. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	**(dev)**	**|E|**	**(dev)**	**|O|**	**(dev)**
Saaltink	0.733	0.	0.846	0.	---	---
Saal.+E-REDUCE	0.719	0.	0.777	0.	---	---
1-REMOVE	---	---	---	---	0.745	0.014
2-REMOVE	---	---	---	---	0.719	0.014
3-REMOVE	---	---	---	---	0.716	0.014
Saal.+E-R.+1-R.	0.698	0.	0.759	0.	0.735	0.011
Saal.+E-R.+2-R.	0.696	0.001	0.757	0.002	0.715	0.014
Saal.+E-R.+3-R.	0.696	0.001	0.757	0.002	0.722	0.025

Table A.12: The relative results of applying different reduction operations on the environment de_vertigo. The columns $|\mathbf{V}|$, $|\mathbf{E}|$ and $|\mathbf{O}|$ list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.13: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing $|\mathbf{V}|$ and Table (b) gives the found significance levels when comparing $|\mathbf{E}|$.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	

Table A.14: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying `d-REMOVE` for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.05	

(c)

Table A.15: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+`E-REDUCE`+`d-REMOVE` for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.4 de_vertigo_scaled

	|V|	**(dev)**	**|E|**	**(dev)**	**|O|**	**(dev)**	**t (s)**	**(dev)**
Original	1196.	---	1480.	---	409.	---	---	---
Saaltink	652.	0.	936.	0.	---	---	0.	0.
Saal.+E-REDUCE	626.	0.	894.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	219.25	7.2	0.13	0.02
2-REMOVE	---	---	---	---	202.75	7.08	0.25	0.02
3-REMOVE	---	---	---	---	191.95	5.78	0.49	0.01
Saal.+E-R.+1-R.	581.05	2.26	849.	2.34	204.	5.34	0.13	0.01
Saal.+E-R.+2-R.	579.	2.55	847.	2.55	189.15	6.11	0.22	0.01
Saal.+E-R.+3-R.	577.75	1.55	845.75	1.55	184.7	5.27	0.42	0.01

Table A.16: The absolute results of applying different reduction operations on the environment de_vertigo_scaled. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.545	0.	0.632	0.	---	---
Saal.+E-REDUCE	0.523	0.	0.604	0.	---	---
1-REMOVE	---	---	---	---	0.536	0.018
2-REMOVE	---	---	---	---	0.496	0.017
3-REMOVE	---	---	---	---	0.469	0.014
Saal.+E-R.+1-R.	0.486	0.002	0.574	0.002	0.499	0.013
Saal.+E-R.+2-R.	0.484	0.002	0.572	0.002	0.462	0.015
Saal.+E-R.+3-R.	0.483	0.001	0.571	0.001	0.452	0.013

Table A.17: The relative results of applying different reduction operations on the environment de_vertigo_scaled. The columns |**V**|, |**E**| and |**O**| list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.18: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing |**V**| and Table (b) gives the found significance levels when comparing |**E**|.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.19: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying d-REMOVE for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of |**O**|.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.05

(c)

Table A.20: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+E-REDUCE+d-REMOVE for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of |**V**| (Table (a)), |**E**| (Table (a)) and |**O**| (Table (a)).

A.5 max

	|V|	(dev)	|E|	(dev)	|O|	(dev)	t (s)	(dev)
Original	5933.	---	8034.	---	112945.	---	---	---
Saaltink	5919.	0.	8020.	0.	---	---	0.	0.
Saal.+E-REDUCE	5919.	0.	8018.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	54235.4	384.19	270.62	11.59
2-REMOVE	---	---	---	---	49381.5	246.42	437.89	25.05
3-REMOVE	---	---	---	---	44859.5	235.79	778.72	58.04
Saal.+E-R.+1-R.	5792.95	6.11	7891.6	6.06	53933.	355.8	291.74	24.76
Saal.+E-R.+2-R.	5758.2	8.35	7856.75	8.48	48827.	198.9	491.38	2.92
Saal.+E-R.+3-R.	5689.25	8.4	7787.75	8.69	44243.4	100.52	913.43	34.12

Table A.21: The absolute results of applying different reduction operations on the environment max. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.998	0.	0.998	0.	---	---
Saal.+E-REDUCE	0.998	0.	0.998	0.	---	---
1-REMOVE	---	---	---	---	0.48	0.003
2-REMOVE	---	---	---	---	0.437	0.002
3-REMOVE	---	---	---	---	0.397	0.002
Saal.+E-R.+1-R.	0.976	0.001	0.982	0.001	0.478	0.003
Saal.+E-R.+2-R.	0.971	0.001	0.978	0.001	0.432	0.002
Saal.+E-R.+3-R.	0.959	0.001	0.969	0.001	0.392	0.001

Table A.22: The relative results of applying different reduction operations on the environment max. The columns $|\mathbf{V}|$, $|\mathbf{E}|$ and $|\mathbf{O}|$ list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.23: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing $|\mathbf{V}|$ and Table (b) gives the found significance levels when comparing $|\mathbf{E}|$.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.24: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying `d-REMOVE` for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.25: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+`E-REDUCE`+`d-REMOVE` for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.6 tf1

	|V|	**(dev)**	**|E|**	**(dev)**	**|O|**	**(dev)**	**t (s)**	**(dev)**
Original	9190.	---	12000.	---	130536.	---	---	---
Saaltink	8337.	0.	11147.	0.	---	---	0.	0.
Saal.+E-REDUCE	8307.	0.	11096.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	57942.6	956.72	438.6	17.43
2-REMOVE	---	---	---	---	50002.1	96.72	687.85	51.21
3-REMOVE	---	---	---	---	47098.8	68.56	1162.83	81.
Saal.+E-R.+1-R.	7834.05	8.86	10622.1	8.86	56736.2	409.94	650.65	48.7
Saal.+E-R.+2-R.	7725.7	7.68	10510.	7.84	49415.5	286.57	892.18	112.63
Saal.+E-R.+3-R.	7651.	8.97	10432.5	9.59	46510.	95.28	1536.14	16.32

Table A.26: The absolute results of applying different reduction operations on the environment tf1. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.907	0.	0.929	0.	---	---
Saal.+E-REDUCE	0.904	0.	0.925	0.	---	---
1-REMOVE	---	---	---	---	0.444	0.007
2-REMOVE	---	---	---	---	0.383	0.001
3-REMOVE	---	---	---	---	0.361	0.001
Saal.+E-R.+1-R.	0.852	0.001	0.885	0.001	0.435	0.003
Saal.+E-R.+2-R.	0.841	0.001	0.876	0.001	0.379	0.002
Saal.+E-R.+3-R.	0.833	0.001	0.869	0.001	0.356	0.001

Table A.27: The relative results of applying different reduction operations on the environment tf1. The columns $|\mathbf{V}|$, $|\mathbf{E}|$ and $|\mathbf{O}|$ list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.28: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing $|\mathbf{V}|$ and Table (b) gives the found significance levels when comparing $|\mathbf{E}|$.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.29: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying d-REMOVE for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.30: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+E-REDUCE+d-REMOVE for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.7 tf2

	|V|	(dev)	|E|	(dev)	|O|	(dev)	t (s)	(dev)
Original	11 574 .	---	15 300 .	---	38 562 .	---	---	---
Saaltink	10 861.	0.	14 587.	0.	---	---	0.	0.
Saal.+E-REDUCE	10 847.	0.	14 558.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	15 916.1	45.28	133.88	5.24
2-REMOVE	---	---	---	---	13 803.2	35.87	195.86	0.7
3-REMOVE	---	---	---	---	12 704.8	31.71	342.24	1.72
Saal.+E-R.+1-R.	9677.85	11.89	13 373.8	11.96	15 320.1	55.73	226.75	9.73
Saal.+E-R.+2-R.	9371.15	10.4	13 064.7	10.6	13 104.1	51.04	321.16	20.7
Saal.+E-R.+3-R.	9179.2	9.79	12 868.8	10.2	11 904.5	35.09	528.67	30.71

Table A.31: The absolute results of applying different reduction operations on the environment tf2. The columns |**V**|, |**E**|, |**O**| and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.938	0.	0.953	0.	---	---
Saal.+E-REDUCE	0.937	0.	0.952	0.	---	---
1-REMOVE	---	---	---	---	0.413	0.001
2-REMOVE	---	---	---	---	0.358	0.001
3-REMOVE	---	---	---	---	0.329	0.001
Saal.+E-R.+1-R.	0.836	0.001	0.874	0.001	0.397	0.001
Saal.+E-R.+2-R.	0.81	0.001	0.854	0.001	0.34	0.001
Saal.+E-R.+3-R.	0.793	0.001	0.841	0.001	0.309	0.001

Table A.32: The relative results of applying different reduction operations on the environment tf2. The columns |**V**|, |**E**| and |**O**| list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.33: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing |**V**| and Table (b) gives the found significance levels when comparing |**E**|.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.34: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying `d-REMOVE` for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.35: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+`E-REDUCE`+`d-REMOVE` for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.8 tf3

	|V|	**(dev)**	**|E|**	**(dev)**	**|O|**	**(dev)**	**t (s)**	**(dev)**
Original	5179.	---	6871.	---	18387.	---	---	---
Saaltink	4768.	0.	6459.	0.	---	---	0.	0.
Saal.+E-REDUCE	4760.	0.	6438.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	8306.3	46.03	31.12	2.79
2-REMOVE	---	---	---	---	7368.65	33.41	49.96	2.73
3-REMOVE	---	---	---	---	6840.45	28.56	98.13	1.05
Saal.+E-R.+1-R.	4310.7	8.74	5981.3	9.12	8059.55	46.3	59.35	2.66
Saal.+E-R.+2-R.	4206.3	7.65	5876.05	7.8	7093.8	27.86	63.18	4.62
Saal.+E-R.+3-R.	4142.5	7.77	5810.3	8.47	6552.4	19.58	132.28	8.45

Table A.36: The absolute results of applying different reduction operations on the environment tf3. The columns $|\mathbf{V}|$, $|\mathbf{E}|$, $|\mathbf{O}|$ and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.921	0.	0.94	0.	---	---
Saal.+E-REDUCE	0.919	0.	0.937	0.	---	---
1-REMOVE	---	---	---	---	0.452	0.003
2-REMOVE	---	---	---	---	0.401	0.002
3-REMOVE	---	---	---	---	0.372	0.002
Saal.+E-R.+1-R.	0.832	0.002	0.871	0.001	0.438	0.003
Saal.+E-R.+2-R.	0.812	0.001	0.855	0.001	0.386	0.002
Saal.+E-R.+3-R.	0.8	0.002	0.846	0.001	0.356	0.001

Table A.37: The relative results of applying different reduction operations on the environment tf3. The columns $|\mathbf{V}|$, $|\mathbf{E}|$ and $|\mathbf{O}|$ list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.38: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing $|\mathbf{V}|$ and Table (b) gives the found significance levels when comparing $|\mathbf{E}|$.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.39: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying d-REMOVE for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.40: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+E-REDUCE+d-REMOVE for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.9 uulib

	|V|	(dev)	|E|	(dev)	|O|	(dev)	t (s)	(dev)
Original	298.	---	421.	---	813.	---	---	---
Saaltink	289.	0.	412.	0.	---	---	0.	0.
Saal.+E-REDUCE	289.	0.	411.	0.	---	---	0.	0.
1-REMOVE	---	---	---	---	391.2	9.38	0.08	0.
2-REMOVE	---	---	---	---	369.9	7.74	0.13	0.01
3-REMOVE	---	---	---	---	346.55	5.65	0.25	0.02
Saal.+E-R.+1-R.	264.3	2.25	386.3	2.25	390.2	7.88	0.09	0.01
Saal.+E-R.+2-R.	257.75	1.55	379.75	1.55	359.2	5.92	0.17	0.02
Saal.+E-R.+3-R.	251.15	0.81	372.8	1.01	338.05	3.9	0.3	0.02

Table A.41: The absolute results of applying different reduction operations on the environment uulib. The columns |**V**|, |**E**|, |**O**| and **t (s)** list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within two decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	|V|	(dev)	|E|	(dev)	|O|	(dev)
Saaltink	0.97	0.	0.979	0.	---	---
Saal.+E-REDUCE	0.97	0.	0.976	0.	---	---
1-REMOVE	---	---	---	---	0.481	0.012
2-REMOVE	---	---	---	---	0.455	0.01
3-REMOVE	---	---	---	---	0.426	0.007
Saal.+E-R.+1-R.	0.887	0.008	0.918	0.005	0.48	0.01
Saal.+E-R.+2-R.	0.865	0.005	0.902	0.004	0.442	0.007
Saal.+E-R.+3-R.	0.843	0.003	0.886	0.002	0.416	0.005

Table A.42: The relative results of applying different reduction operations on the environment uulib. The columns |**V**|, |**E**| and |**O**| list the average results for each experiment. The columns with the header **(dev)** give the standard deviation of the attribute in the previous column. All results are given within three decimal places. '—' signifies that that particular set of operations did not effect that particular attribute.

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	

(a)

	Original	Saaltink
Saaltink	0.01	
Saal.+E-REDUCE	0.01	0.01

(b)

Table A.43: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink and Saaltink+E-REDUCE. Table (a) gives the found significance levels when comparing |**V**| and Table (b) gives the found significance levels when comparing |**E**|.

	Original	1-REMOVE	2-REMOVE
1-REMOVE	0.01		
2-REMOVE	0.01	0.01	
3-REMOVE	0.01	0.01	0.01

Table A.44: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying `d-REMOVE` for $d \in \{1, 2, 3\}$. This table reports the found significance level when comparing the reduction of $|\mathbf{O}|$.

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(a)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(b)

	Orig	1-REM	2-REM
1-REM	0.01		
2-REM	0.01	0.01	
3-REM	0.01	0.01	0.01

(c)

Table A.45: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the original environment to the environments obtained after applying Saaltink+`E-REDUCE`+`d-REMOVE` for $d \in \{1, 2, 3\}$. These tables report the found significance level when comparing the reduction of $|\mathbf{V}|$ (Table (a)), $|\mathbf{E}|$ (Table (a)) and $|\mathbf{O}|$ (Table (a)).

A.10 Reduced environments

	|V|	**|E|**	**EW**	**|O|**	**|TF|**	**O/V**
as_oilrig_min	496	798	816	892	2	1.8
as_oilrig_scaled_min	1245	1562	1562	1888	4	1.52
de_vertigo_min	255	485	518	26	4	0.1
de_vertigo_scaled_min	575	843	845	175	5	0.3
max_min	5679	7777	7779	44 092	1	7.76
tf1_min	7648	10 430	10 436	46 345	14	6.06
tf2_min	9161	12 849	12 862	11 842	46	1.29
tf3_min	4144	5813	5824	6574	11	1.59
uulib_min	252	374	375	329	1	1.31

Table A.46: Statistics of the reduced environments used for further experimentation. The column **EW** contains the sum of all edge capacities for that environment.

Appendix B

SPH results

	|T|	(dev)	Johnson	(dev)	t (s)	(dev)
as_oilrig min.	20.5	1.67	0.04	0.01	0.07	0.
as_oilrig or.	25.2	0.62	0.06	0.01	0.1	0.01
as_oilrig_scaled min.	93.95	0.83	0.24	0.	0.58	0.01
as_oilrig_scaled or.	98.85	1.23	0.41	0.	7.53	0.01
de_vertigo min.	6.15	0.37	0.01	0.	0.01	0.01
de_vertigo or.	6.45	0.51	0.02	0.	0.02	0.
de_vertigo_scaled min.	15.	0.	0.06	0.	0.06	0.
de_vertigo_scaled or.	14.	0.	0.19	0.01	0.28	0.
max min.	113.55	1.76	6.03	0.06	36.51	0.17
max or.	119.1	1.59	6.34	0.08	275.66	0.54
tf1 min.	1185.8	8.21	10.77	0.08	592.63	109.22
tf1 or.	1183.	8.31	14.43	0.11	7316.26	7.14
tf2 min.	1222.65	8.54	13.98	0.1	21.31	0.09
tf2 or.	1276.95	6.8	20.69	0.21	53.52	4.14
tf3 min.	702.25	8.94	2.95	0.03	13.71	1.87
tf3 or.	707.95	6.74	4.29	0.02	113.35	22.19
uulib min.	10.2	1.01	0.01	0.	0.02	0.
uulib or.	13.6	0.5	0.02	0.01	0.04	0.

Table B.1: The results for the shortest path heuristic. The column **Johnson** lists the time spent in Johnson's all pair shortest path algortihms, while **t (s)** lists the total running time.

Appendix C

HH results

C.1 Results for each phase

	|T| C	(dev)	t (s)	(dev)	|T| M	(dev)	t (s)	(dev)	|T| R	(dev)	t (s)	(dev)
as_oilrig min.	67.	0.	0.	0.	24.6	1.27	0.	0.	19.8	0.41	0.01	0.
as_oilrig or.	61.	0.	0.	0.	23.6	1.19	0.	0.	19.45	0.51	0.01	0.
as_oilrig_scaled min.	66.	0.	0.01	0.01	31.7	0.66	0.	0.	30.7	0.66	0.02	0.01
as_oilrig_scaled or.	84.	0.	0.01	0.	29.9	0.91	0.	0.	28.8	0.77	0.04	0.
de_vertigo min.	4.	0.	0.	0.	4.	0.	0.	0.	2.	0.	0.	0.
de_vertigo or.	4.	0.	0.	0.	4.	0.	0.	0.	2.	0.	0.	0.
de_vertigo_scaled min.	5.	0.	0.	0.	2.	0.	0.	0.	2.	0.	0.	0.
de_vertigo_scaled or.	4.	0.	0.	0.	2.	0.	0.	0.	2.	0.	0.01	0.
max min.	28.	0.	0.03	0.01	28.	0.	0.	0.01	28.	0.	0.09	0.01
max or.	28.	0.	0.06	0.	28.	0.	0.	0.	28.	0.	0.15	0.
tf1 min.	161.	0.	0.05	0.	93.4	1.05	0.01	0.	78.	0.	0.36	0.07
tf1 or.	166.	0.	0.08	0.01	74.7	0.8	0.02	0.	72.4	0.5	0.62	0.12
tf2 min.	1494.	0.	0.06	0.	490.	22.68	0.02	0.	348.9	1.86	2.69	0.56
tf2 or.	1780.	0.	0.09	0.01	486.	24.74	0.03	0.	340.65	3.01	3.46	0.9
tf3 min.	547.	0.	0.03	0.01	173.15	3.63	0.01	0.	155.6	0.75	0.44	0.09
tf3 or.	635.	0.	0.03	0.	171.15	3.36	0.01	0.	156.75	1.29	0.56	0.15
uulib min.	11.	0.	0.	0.	9.	0.	0.	0.	9.	0.	0.	0.
uulib or.	12.	0.	0.	0.	9.	0.	0.	0.	9.	0.	0.	0.

Table C.1: This table lists the results of the HH experiments after each different phase. In column $|T|C$ and the three following columns the results for the cluster and component creation phase are given. $|T|M$ and subsequent columns show information about the merge phase. The last four columns show the results for the redistribution phase.

C.2 Relative change for each phase

	|T| C	|T| M	Change	|T| R	Change
as_oilrig min.	67.	24.6	42.4	19.8	4.8
as_oilrig or.	61.	23.6	37.4	19.45	4.15
as_oilrig_scaled min.	66.	31.7	34.3	30.7	1.
as_oilrig_scaled or.	84.	29.9	54.1	28.8	1.1
de_vertigo min.	4.	4.	0.	2.	2.
de_vertigo or.	4.	4.	0.	2.	2.
de_vertigo_scaled min.	5.	2.	3.	2.	0.
de_vertigo_scaled or.	4.	2.	2.	2.	0.
max min.	28.	28.	0.	28.	0.
max or.	28.	28.	0.	28.	0.
tf1 min.	161.	93.4	67.6	78.	15.4
tf1 or.	166.	74.7	91.3	72.4	2.3
tf2 min.	1494.	490.	1004.	348.9	141.1
tf2 or.	1780.	486.	1294.	340.65	145.35
tf3 min.	547.	173.15	373.85	155.6	17.55
tf3 or.	635.	171.15	463.85	156.75	14.4
uulib min.	11.	9.	2.	9.	0.
uulib or.	12.	9.	3.	9.	0.

Table C.2: This table show the average change after each phase.

C.3 Significance of change for each phase

C.3.1 as_oilrig

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01	0.01	0.01	0.01

Table C.3: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.2 as_oilrig_scaled

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01	0.01	0.01	0.01

Table C.4: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.3 de_vertigo

	Original		Minimized	
	C	M	C	M
M				
R	0.01	0.01	0.01	0.01

Table C.5: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.4 de_vertigo_scaled

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01		0.01	

Table C.6: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.5 max

	Original		Minimized	
	C	M	C	M
M				
R				

Table C.7: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.6 tf1

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01	0.01	0.01	0.01

Table C.8: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.7 tf2

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01	0.01	0.01	0.01

Table C.9: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.8 tf3

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01	0.01	0.01	0.01

Table C.10: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

C.3.9 uulib

	Original		Minimized	
	C	M	C	M
M	0.01		0.01	
R	0.01		0.01	

Table C.11: Results for the Anova significance test using Tukey post-hoc analysis, comparing the relevance of the change in score after each phase in the HH algorithm.

Appendix D

Genetic results

D.1 as_oilrig

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	4.	22.	1.83	18.	6.63	924.3	159.73
Def. Orig.	4.	22.25	1.71	18.33	6.41	1516.47	263.5
Incr. mut. Min.	6.	28.	4.47	28.	4.47	1436.72	248.23
Incr. mut. Orig.	5.	35.8	3.96	33.	7.72	1758.48	101.79
Def.+B0.3 Min.	6.	20.83	2.48	20.83	2.48	1800.03	0.03
Def.+B0.3 Orig.	5.	20.4	0.55	18.33	5.09	1723.04	188.58
Def.+B0.5 Min.	4.	21.5	1.29	15.83	9.28	1652.	362.59
Def.+B0.5 Orig.	5.	21.2	3.42	19.5	5.17	1656.29	352.11

Table D.1: This table describes the result for the genetic algorithm experiments done on environment as_oilrig. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.2 as_oilrig_scaled

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	0.	---	---	262.5	173.35	1800.22	0.27
Def. Orig.	0.	---	---	436.83	175.51	1800.36	0.38
Incr. mut. Min.	3.	410.33	38.84	359.17	145.67	1800.42	0.48
Incr. mut. Orig.	2.	561.5	62.93	400.	241.11	1800.78	0.62
Def.+B0.3 Min.	0.	---	---	230.5	121.55	1800.37	0.48
Def.+B0.3 Orig.	0.	---	---	480.33	137.03	1800.84	0.85
Def.+B0.5 Min.	1.	281.	0.	302.5	61.19	1800.31	0.32
Def.+B0.5 Orig.	0.	---	---	407.	200.95	1801.1	1.09

Table D.2: This table describes the result for the genetic algorithm experiments done on environment as_oilrig_scaled. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.3 de_vertigo

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	2.	6.5	0.71	4.83	1.72	19.38	4.87
Def. Orig.	4.	8.25	0.96	7.33	1.63	25.57	2.88
Incr. mut. Min.	4.	9.	3.46	7.67	3.44	25.69	12.05
Incr. mut. Orig.	3.	11.67	2.52	8.	4.77	33.44	10.27
Def.+B0.3 Min.	6.	6.	0.	6.	0.	1800.	0.
Def.+B0.3 Orig.	6.	6.	0.	6.	0.	1800.	0.
Def.+B0.5 Min.	5.	6.	0.	5.5	1.22	1501.79	730.46
Def.+B0.5 Orig.	4.	6.	0.	4.83	2.4	1800.	0.

Table D.3: This table describes the result for the genetic algorithm experiments done on environment de_vertigo. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.4 de_vertigo_scaled

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	6.	15.33	2.34	15.33	2.34	294.17	80.8
Def. Orig.	4.	16.75	2.22	14.5	4.32	999.87	290.3
Incr. mut. Min.	5.	20.8	3.9	19.5	4.72	391.74	122.57
Incr. mut. Orig.	5.	30.6	5.98	28.67	7.15	1546.56	317.32
Def.+B0.3 Min.	6.	14.5	1.64	14.5	1.64	1800.01	0.01
Def.+B0.3 Orig.	6.	13.67	0.82	13.67	0.82	1800.04	0.06
Def.+B0.5 Min.	3.	13.33	1.15	9.	4.82	1643.19	384.13
Def.+B0.5 Orig.	5.	15.	2.35	12.83	5.71	1800.06	0.06

Table D.4: This table describes the result for the genetic algorithm experiments done on environment de_vertigo_scaled. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.5 max

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	0.	---	---	2681.	574.	1803.21	3.01
Def. Orig.	0.	---	---	1904.83	1015.69	1802.81	2.31
Incr. mut. Min.	0.	---	---	1641.5	773.85	1803.12	2.24
Incr. mut. Orig.	0.	---	---	2107.5	905.6	1804.13	2.66
Def.+B0.3 Min.	0.	---	---	1763.33	930.3	1801.98	1.35
Def.+B0.3 Orig.	0.	---	---	1286.17	969.13	1801.54	1.05
Def.+B0.5 Min.	0.	---	---	1784.	1006.66	1803.22	2.26
Def.+B0.5 Orig.	0.	---	---	1911.17	1219.48	1803.27	1.38

Table D.5: This table describes the result for the genetic algorithm experiments done on environment max. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.6 tf1

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	0.	---	---	3541.67	1346.77	1807.92	10.45
Def. Orig.	0.	---	---	3093.	1773.62	1819.41	14.39
Incr. mut. Min.	0.	---	---	3736.33	734.61	1807.34	8.07
Incr. mut. Orig.	0.	---	---	3252.83	1594.49	1813.27	9.18
Def.+B0.3 Min.	0.	---	---	3328.67	985.08	1809.21	7.22
Def.+B0.3 Orig.	0.	---	---	4451.	601.32	1814.68	12.47
Def.+B0.5 Min.	0.	---	---	3447.	1269.62	1809.2	10.55
Def.+B0.5 Orig.	0.	---	---	3065.	916.66	1803.99	6.67

Table D.6: This table describes the result for the genetic algorithm experiments done on environment tf1. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.7 tf2

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	0.	---	---	4479.33	873.21	1815.63	24.19
Def. Orig.	0.	---	---	4097.33	1655.54	1821.34	28.08
Incr. mut. Min.	0.	---	---	4116.5	1651.07	1814.07	15.61
Incr. mut. Orig.	0.	---	---	4649.83	1419.57	1822.19	19.82
Def.+B0.3 Min.	0.	---	---	3244.5	1594.77	1818.38	11.95
Def.+B0.3 Orig.	0.	---	---	2985.67	2017.91	1828.5	34.64
Def.+B0.5 Min.	0.	---	---	4581.83	537.62	1809.02	9.07
Def.+B0.5 Orig.	0.	---	---	4447.	1209.25	1824.87	32.76

Table D.7: This table describes the result for the genetic algorithm experiments done on environment tf2. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.8 tf3

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	0.	---	---	1828.67	316.69	1805.24	4.25
Def. Orig.	0.	---	---	1952.83	1307.68	1807.38	6.25
Incr. mut. Min.	0.	---	---	1763.67	770.08	1803.46	3.18
Incr. mut. Orig.	0.	---	---	2142.83	703.44	1810.22	7.85
Def.+B0.3 Min.	0.	---	---	1958.	444.11	1806.74	3.98
Def.+B0.3 Orig.	0.	---	---	2152.33	850.47	1804.47	4.27
Def.+B0.5 Min.	0.	---	---	1625.33	887.56	1805.53	3.7
Def.+B0.5 Orig.	0.	---	---	1601.	940.25	1807.26	4.15

Table D.8: This table describes the result for the genetic algorithm experiments done on environment tf3. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

D.9 uulib

	Valid	Valid Cut	(dev)	Cuts	(dev)	t (s)	(dev)
Def. Min.	3.	10.	1.	8.5	2.81	183.98	68.88
Def. Orig.	5.	9.8	1.1	9.	2.19	244.34	74.22
Incr. mut. Min.	2.	11.5	3.54	8.67	3.93	163.62	46.19
Incr. mut. Orig.	5.	13.2	2.17	12.17	3.19	336.26	56.31
Def.+B0.3 Min.	5.	9.4	0.89	8.17	3.13	1800.	0.
Def.+B0.3 Orig.	4.	9.	0.	7.83	2.4	1800.01	0.01
Def.+B0.5 Min.	5.	10.4	1.34	9.83	1.83	1528.88	664.11
Def.+B0.5 Orig.	6.	9.83	1.33	9.83	1.33	1800.01	0.01

Table D.9: This table describes the result for the genetic algorithm experiments done on environment uulib. In the column '**Valid**' the number of runs resulting in a valid decomposition (i.e. a residual flow of 0) of the environment is listed. The column '**Valid Cut**' lists the average value of all valid cuts. Under '**Cuts**' the average number of cuts of all best solutions (i.e. minimal residual flow) is listed.

Appendix E

LS results

E.1 as_oilrig

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	15.	0.	4.17	3.17	990.6	330.18	60.56	13.93	11 990.6	330.18
Tabu 10	15.	0.	4.01	3.1	754.2	159.99	229.03	21.03	10 754.2	159.99
Sim.	27.7	3.83	3.16	2.01	8354.3	4012.75	6.49	1.97	18 354.3	4012.75
Sim. L	15.2	0.42	0.18	0.06	936.	334.01	1.95	0.1	10 936.	334.01
Sim. I	15.	0.	0.21	0.06	1117.1	327.84	1.93	0.13	11 117.1	327.84
Sim. Tmin	16.7	1.77	10.66	6.8	16 561.9	9595.81	16.64	7.07	26 561.9	9595.81
Sim. Tmin L	15.	0.	0.21	0.09	1100.4	529.05	1.95	0.13	11 100.4	529.05
Sim. Tmin I	15.	0.	0.18	0.04	903.7	200.35	1.92	0.11	10 903.7	200.35

Table E.1: Results of the different local search tests on environment as_oilrig. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.							
Sim.L	0.1	0.1					
Sim.I	0.1	0.1					
Sim.T	0.01	0.01	0.01	0.01	0.01		
Sim.T.L	0.1	0.1				0.01	
Sim.T.I	0.1	0.1				0.01	

(b)

Table E.2: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.2 as_oilrig_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	25.	4.22	1.05	0.52	612.	88.7	19.75	3.47	11612.	88.7
Tabu 10	19.2	1.93	2.52	1.63	588.7	109.36	61.31	15.98	10588.7	109.36
Sim.	46.	2.98	1.35	0.66	6115.6	2422.48	3.38	0.59	16115.6	2422.48
Sim. L	32.8	0.42	0.14	0.05	900.5	294.31	1.59	0.13	10900.5	294.31
Sim. I	33.	0.	0.13	0.07	879.2	563.26	1.55	0.05	10879.2	563.26
Sim. Tmin	37.8	2.57	2.84	1.68	8812.5	5296.84	5.75	1.37	18812.5	5296.84
Sim. Tmin L	32.8	0.63	0.13	0.06	813.1	379.11	1.61	0.16	10813.1	379.11
Sim. Tmin I	32.9	0.32	0.12	0.03	760.1	233.3	1.55	0.1	10760.1	233.3

Table E.3: Results of the different local search tests on environment as_oilrig_min. |**T**| is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.	0.01	0.01					
Sim.L	0.01	0.01	0.01				
Sim.I	0.01	0.01	0.01				
Sim.T	0.01	0.01	0.01	0.01	0.01		
Sim.T.L	0.01	0.01	0.01			0.01	
Sim.T.I	0.01	0.01	0.01			0.01	

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.		0.1					
Sim.L		0.01	0.1				
Sim.I		0.01	0.1				
Sim.T	0.01		0.01	0.01	0.01		
Sim.T.L		0.01	0.1			0.01	
Sim.T.I		0.01	0.05			0.01	

(b)

Table E.4: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.3 as_oilrig_scaled

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	37.1	1.52	144.63	65.93	7736.2	2895.39	285.37	44.3	18736.2	2895.39
Tabu 10	40.2	3.12	190.09	192.7	4672.6	2997.6	822.51	198.1	14672.6	2997.6
Sim.	92.6	11.34	9.81	5.68	18898.9	7736.03	15.52	5.66	28898.9	7736.03
Sim. L	42.5	2.64	1.47	0.26	2910.6	812.63	4.58	0.37	12910.6	812.63
Sim. I	41.6	2.01	1.44	0.15	2874.	469.45	4.52	0.16	12874.	469.45
Sim. Tmin	39.3	2.91	90.82	37.65	33006.8	8452.36	132.81	41.04	43006.8	8452.36
Sim. Tmin L	43.5	2.32	1.34	0.08	2515.4	268.13	4.5	0.18	12515.4	268.13
Sim. Tmin I	42.2	2.35	1.82	0.83	4070.2	2615.89	4.96	0.89	14070.2	2615.89

Table E.5: Results of the different local search tests on environment as_oilrig_scaled. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L	0.1		0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.01	0.01					
Sim.T		0.1					
Sim.T.L	0.01	0.01					
Sim.T.I	0.01	0.01					

(b)

Table E.6: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.4 as_oilrig_scaled_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	37.9	3.07	17.94	8.39	3168.9	1082.96	53.43	4.51	14168.9	1082.96
Tabu 10	35.8	2.1	72.6	43.57	4617.7	2183.79	150.68	23.76	14617.7	2183.79
Sim.	92.3	5.96	3.82	1.48	15015.1	6352.96	6.32	1.49	25015.1	6352.96
Sim. L	41.3	2.	0.78	0.24	2575.9	1164.88	2.57	0.36	12575.9	1164.88
Sim. I	42.5	4.14	0.76	0.31	2430.4	1534.23	2.57	0.37	12430.4	1534.23
Sim. Tmin	43.4	4.22	18.28	9.93	24511.9	9837.11	28.12	10.47	34511.9	9837.11
Sim. Tmin L	40.3	3.16	0.99	0.53	3722.	2847.94	2.74	0.58	13722.	2847.94
Sim. Tmin I	42.1	3.78	0.7	0.16	2171.1	843.71	2.45	0.25	12171.1	843.71

Table E.7: Results of the different local search tests on environment as_oilrig_scaled_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L		0.05	0.01				
Sim.I		0.01	0.01				
Sim.T	0.05	0.01	0.01				
Sim.T.L			0.01				
Sim.T.I		0.01	0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.		0.01					
Sim.L		0.01					
Sim.I		0.01					
Sim.T		0.01					
Sim.T.L		0.01					
Sim.T.I		0.01					

(b)

Table E.8: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.5 de_vertigo

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	2.	0.	0.89	0.47	403.7	9.37	542.36	95.86	11 403.7	9.37
Tabu 10	2.	0.	1.94	1.13	372.4	5.89	1560.45	19.34	10 372.4	5.89
Sim.	2.	0.	2.34	1.3	1793.	458.98	39.49	4.62	11 793.	458.98
Sim. L	2.	0.	0.04	0.01	393.3	20.07	2.08	0.02	10 393.3	20.07
Sim. I	2.	0.	0.04	0.	382.	6.57	2.07	0.03	10 382.	6.57
Sim. Tmin	2.	0.	0.82	0.57	1435.	90.2	69.84	9.35	11 435.	90.2
Sim. Tmin L	2.	0.	0.04	0.	386.4	12.11	2.07	0.02	10 386.4	12.11
Sim. Tmin I	2.	0.	0.04	0.01	389.3	13.31	2.05	0.01	10 389.3	13.31

Table E.9: Results of the different local search tests on environment de_vertigo. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.							
Sim.L							
Sim.I							
Sim.T							
Sim.T.L							
Sim.T.I							

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.02						
Sim.	0.01						
Sim.L		0.01	0.01				
Sim.I	0.1	0.01	0.01				
Sim.T		0.01	0.01				
Sim.T.L	0.1	0.01	0.01				
Sim.T.I	0.1	0.01	0.01				

(b)

Table E.10: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.6 de_vertigo_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	11.	3.46	0.34	0.16	277.6	8.88	59.22	10.37	11 277.6	8.88
Tabu 10	6.9	2.56	0.68	0.42	258.3	8.34	136.15	10.17	10 258.3	8.34
Sim.	30.3	2.31	3.98	3.63	5112.2	4683.94	10.51	2.73	15 112.2	4683.94
Sim. L	25.5	9.68	0.02	0.	261.9	5.88	1.49	0.08	10 261.9	5.88
Sim. I	22.2	10.4	0.02	0.	260.8	3.68	1.51	0.01	10 260.8	3.68
Sim. Tmin	29.6	6.55	2.9	3.22	2711.5	2149.1	15.13	2.96	12 711.5	2149.1
Sim. Tmin L	16.2	9.74	0.02	0.	264.	4.19	1.51	0.01	10 264.	4.19
Sim. Tmin I	19.8	6.75	0.02	0.	260.7	5.08	1.51	0.01	10 260.7	5.08

Table E.11: Results of the different local search tests on environment de_vertigo_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.02	0.01					
Sim.T	0.01	0.01					
Sim.T.L		0.1	0.01	0.1		0.01	
Sim.T.I		0.01	0.05			0.1	

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T	0.05	0.1		0.01	0.01		
Sim.T.L			0.01			0.01	
Sim.T.I			0.01			0.01	

(b)

Table E.12: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.7 de_vertigo_scaled

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	2.	0.	42.3	32.78	1386.9	64.95	1181.04	124.28	12 386.9	64.95
Tabu 10	2.4	1.26	150.16	221.38	1362.8	259.02	1801.57	1.35	3723.2	492.03
Sim.	2.	0.	39.09	31.89	5884.8	1464.54	243.41	39.64	15 884.8	1464.54
Sim. L	3.2	1.55	0.41	0.02	1252.8	32.97	5.49	0.64	11 252.8	32.97
Sim. I	4.6	1.84	0.41	0.01	1241.7	9.51	5.47	0.74	11 241.7	9.51
Sim. Tmin	2.	0.	26.77	24.31	4525.7	493.08	435.09	45.84	14 525.7	493.08
Sim. Tmin L	3.9	1.66	0.41	0.01	1252.1	31.69	5.39	0.61	11 252.1	31.69
Sim. Tmin I	3.5	1.58	0.42	0.01	1255.7	20.46	5.56	0.8	11 255.7	20.46

Table E.13: Results of the different local search tests on environment de_vertigo_scaled. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.							
Sim.L							
Sim.I	0.01	0.01	0.01				
Sim.T					0.01		
Sim.T.L	0.05		0.05			0.05	
Sim.T.I							

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.1						
Sim.		0.1					
Sim.L		0.01					
Sim.I		0.01					
Sim.T		0.05					
Sim.T.L		0.01					
Sim.T.I		0.01					

(b)

Table E.14: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.8 de_vertigo_scaled_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	3.6	0.7	10.53	15.42	754.2	170.51	117.78	13.67	11754.2	170.51
Tabu 10	3.6	0.7	4.16	2.44	588.4	8.68	187.34	16.03	10588.4	8.68
Sim.	4.	0.	4.62	3.5	2842.5	451.05	38.25	4.2	12842.5	451.05
Sim. L	6.2	1.99	0.1	0.	608.2	10.93	2.05	0.3	10608.2	10.93
Sim. I	4.9	1.29	0.1	0.	604.3	10.89	2.12	0.28	10604.3	10.89
Sim. Tmin	4.	0.	3.1	2.99	2464.7	237.96	49.33	6.26	12464.7	237.96
Sim. Tmin L	5.6	1.78	0.1	0.	609.	13.31	2.23	0.23	10609.	13.31
Sim. Tmin I	4.9	2.13	0.1	0.	611.7	13.62	2.12	0.29	10611.7	13.62

Table E.15: Results of the different local search tests on environment de_vertigo_scaled_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.							
Sim.L	0.01	0.01	0.02				
Sim.I							
Sim.T				0.02			
Sim.T.L	0.05	0.05					
Sim.T.I							

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.							
Sim.L	0.01						
Sim.I	0.01						
Sim.T	0.1						
Sim.T.L	0.01						
Sim.T.I	0.01						

(b)

Table E.16: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.9 max

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	71.5	0.85	385.32	301.15	9712.9	3246.89	1373.39	240.87	20 354.4	2522.62
Tabu 10	72.	0.	529.82	382.85	8054.1	2022.58	1800.22	0.15	14 814.7	153.3
Sim.	200.5	30.01	72.7	27.55	30 432.3	4263.6	135.59	31.1	40 432.3	4263.6
Sim. L	72.1	0.32	22.44	5.04	9052.5	2302.39	44.58	5.11	19 052.5	2302.39
Sim. I	72.3	0.48	26.01	6.02	10 645.3	2750.64	48.46	5.98	20 645.3	2750.64
Sim. Tmin	72.3	0.67	249.44	50.29	34 380.9	2633.42	432.2	53.09	44 380.9	2633.42
Sim. Tmin L	72.1	0.32	27.	6.54	11 123.7	2975.7	49.15	6.68	21 123.7	2975.7
Sim. Tmin I	72.1	0.32	22.27	5.64	8939.3	2567.53	44.36	5.71	18 939.3	2567.53

Table E.17: Results of the different local search tests on environment max. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.01	0.01					
Sim.T		0.02		0.1	0.1		
Sim.T.L	0.01	0.01				0.1	
Sim.T.I	0.01	0.01				0.1	

(b)

Table E.18: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.10 max_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	73.9	1.85	532.02	562.04	12129.5	7160.7	1263.8	334.12	21587.5	4108.98
Tabu 10	74.4	0.7	470.46	246.95	7931.3	1580.25	1799.41	2.3	16311.4	207.23
Sim.	211.9	16.86	69.68	28.71	31707.5	5524.19	125.72	31.63	41707.5	5524.19
Sim. L	74.	0.	16.75	1.6	8732.4	1290.76	29.25	1.6	18732.4	1290.76
Sim. I	74.3	0.67	17.88	3.7	9648.9	2990.17	30.61	3.68	19648.9	2990.17
Sim. Tmin	75.2	0.63	218.63	55.75	33878.	4935.92	374.83	53.06	43878.	4935.92
Sim. Tmin L	74.	0.	18.3	2.98	10022.	2451.73	30.61	3.	20022.	2451.73
Sim. Tmin I	73.8	0.79	15.31	1.3	7559.2	1066.28	27.77	1.36	17559.2	1066.28

Table E.19: Results of the different local search tests on environment max_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.01	0.01					
Sim.T	0.05						
Sim.T.L	0.01	0.01					
Sim.T.I	0.01	0.01					

(b)

Table E.20: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.11 tf1

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	756.9	6.56	353.44	51.04	18 991.6	3129.95	534.45	56.49	29 991.6	3129.95
Tabu 10	743.	6.83	913.33	404.58	21 988.9	8492.61	1339.76	366.41	31 328.2	7540.52
Sim.	1791.	86.3	58.21	5.73	67 063.6	13 532.5	61.96	5.72	77 063.6	13 532.5
Sim. L	744.4	6.92	45.14	3.59	24 728.	5967.35	51.4	3.56	34 728.	5967.35
Sim. I	743.3	6.73	47.46	2.14	28 059.9	3482.19	53.64	2.12	38 059.9	3482.19
Sim. Tmin	738.	9.88	126.73	35.67	62 301.	8574.77	162.64	39.66	72 301.	8574.77
Sim. Tmin L	743.9	5.38	47.67	3.45	28 034.3	5342.43	53.91	3.46	38 034.3	5342.43
Sim. Tmin I	746.3	7.87	46.44	3.83	26 564.6	6227.09	52.72	3.8	36 564.6	6227.09

Table E.21: Results of the different local search tests on environment tf1. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.01	0.01					
Sim.T	0.02	0.01					
Sim.T.L	0.01	0.01					
Sim.T.I	0.01	0.01					

(b)

Table E.22: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.12 tf1_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	739.8	7.9	154.46	10.86	13 347.8	1665.62	208.84	16.17	24 347.8	1665.62
Tabu 10	689.9	7.52	505.19	219.1	27 693.4	14 728.1	642.19	202.65	37 693.4	14 728.1
Sim.	1684.4	95.02	30.93	3.38	61 079.6	16 697.9	32.66	3.29	71 079.6	16 697.9
Sim. L	705.7	7.7	29.08	2.18	23 643.3	5735.75	33.02	2.13	33 643.3	5735.75
Sim. I	707.	4.19	29.96	2.74	25 845.6	6799.23	33.92	2.79	35 845.6	6799.23
Sim. Tmin	690.4	5.68	52.31	12.23	59 869.	9869.68	63.66	12.03	69 869.	9869.68
Sim. Tmin L	706.4	6.42	29.51	2.48	24 484.1	6212.95	33.5	2.51	34 484.1	6212.95
Sim. Tmin I	705.2	7.96	28.51	2.63	22 001.1	6542.66	32.47	2.7	32 001.1	6542.66

Table E.23: Results of the different local search tests on environment tf1_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.05						
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T	0.05		0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.	0.02	0.01					
Sim.L	0.02	0.01					
Sim.I	0.02	0.01					
Sim.T	0.1	0.01					
Sim.T.L	0.02	0.01					
Sim.T.I	0.02	0.01					

(b)

Table E.24: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.13 tf2

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	726.7	12.09	1475.91	213.07	21 097.1	3151.73	1831.93	29.31	22 441.6	2877.07
Tabu 10	710.6	13.21	1616.99	153.88	14 623.4	533.54	1825.	22.36	15 234.1	670.06
Sim.	2020.1	75.62	71.77	10.7	73 711.8	12 856.6	80.4	11.29	83 711.8	12 856.6
Sim. L	711.	8.39	58.84	4.16	28 812.6	6326.62	64.99	4.2	38 812.6	6326.62
Sim. I	711.	8.59	58.68	5.05	28 442.	8597.97	64.84	4.95	38 442.	8597.97
Sim. Tmin	716.9	14.56	894.99	366.35	66 806.	6083.07	1309.55	446.25	76 557.4	6184.19
Sim. Tmin L	702.5	8.21	59.05	4.22	28 917.6	6539.98	65.19	4.29	38 917.6	6539.98
Sim. Tmin I	713.1	11.23	58.03	2.33	27 227.1	3711.68	64.23	2.37	37 227.1	3711.68

Table E.25: Results of the different local search tests on environment tf2. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.01	0.01					
Sim.T	0.01	0.01	0.01	0.01	0.01		
Sim.T.L	0.01	0.01				0.01	
Sim.T.I	0.01	0.01				0.01	

(b)

Table E.26: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.14 tf2_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	724.2	8.43	1207.36	398.9	20 405.4	3608.65	1809.28	53.49	26 831.1	3256.48
Tabu 10	700.4	12.15	1498.68	183.92	13 585.1	1397.66	1824.28	19.7	14 636.6	1288.47
Sim.	1955.6	128.64	47.12	8.45	70 774.3	14 790.3	54.58	11.07	80 774.3	14 790.3
Sim. L	710.5	10.38	37.22	2.11	25 568.2	4377.55	41.98	2.16	35 568.2	4377.55
Sim. I	716.5	11.87	36.73	3.5	24 813.5	7502.89	41.45	3.5	34 813.5	7502.89
Sim. Tmin	710.4	15.34	613.22	201.72	62 049.4	6802.13	834.94	192.64	72 049.4	6802.13
Sim. Tmin L	714.3	11.98	36.64	2.66	24 508.4	5584.18	41.37	2.64	34 508.4	5584.18
Sim. Tmin I	711.6	8.26	37.89	3.47	27 126.1	7465.69	42.55	3.51	37 126.1	7465.69

Table E.27: Results of the different local search tests on environment tf2_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.	0.01	0.01					
Sim.L	0.01	0.01					
Sim.I	0.01	0.01					
Sim.T	0.01	0.01	0.01	0.01	0.01		
Sim.T.L	0.01	0.01				0.01	
Sim.T.I	0.01	0.01				0.01	

(b)

Table E.28: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.15 tf3

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	366.9	7.82	171.83	39.7	15 453.8	2750.92	283.22	39.13	26 453.8	2750.92
Tabu 10	363.3	6.07	799.4	455.38	30 454.5	14 639.6	1049.04	393.28	39 496.3	12 700.4
Sim.	893.6	47.02	16.93	2.86	43 290.	8285.55	20.49	2.9	53 290.	8285.55
Sim. L	377.9	7.92	12.15	0.72	12 877.5	2433.08	15.04	0.77	22 877.5	2433.08
Sim. I	373.4	7.9	12.4	1.24	13 684.4	4197.84	15.32	1.25	23 684.4	4197.84
Sim. Tmin	358.6	8.24	58.42	17.75	41 334.5	6092.57	86.14	18.63	51 334.5	6092.57
Sim. Tmin L	376.2	10.54	11.93	0.72	12 170.	2509.2	14.82	0.75	22 170.	2509.2
Sim. Tmin I	376.4	7.68	11.71	0.44	11 266.7	1530.7	14.6	0.44	21 266.7	1530.7

Table E.29: Results of the different local search tests on environment tf3. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.		0.01					
Sim.L		0.01					
Sim.I		0.01					
Sim.T		0.01					
Sim.T.L		0.01					
Sim.T.I		0.01					

(b)

Table E.30: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.16 tf3_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	383.7	5.52	86.16	26.83	12 607.7	4874.36	147.14	16.12	23 607.7	4874.36
Tabu 10	356.4	11.73	488.5	329.42	29 710.7	23 415.3	657.46	301.28	39 710.7	23 415.3
Sim.	909.8	50.03	11.64	1.84	42 838.3	10 887.1	13.53	1.82	52 838.3	10 887.1
Sim. L	379.	7.12	8.42	0.8	14 103.	3703.21	10.61	0.81	24 103.	3703.21
Sim. I	380.2	11.68	7.8	0.7	11 447.2	3286.14	9.98	0.7	21 447.2	3286.14
Sim. Tmin	363.9	15.94	39.16	6.17	42 373.3	5183.	53.96	8.48	52 373.3	5183.
Sim. Tmin L	379.6	15.19	7.7	0.44	10 730.8	2123.18	9.88	0.42	20 730.8	2123.18
Sim. Tmin I	380.6	11.06	8.14	0.65	12 877.7	2963.02	10.29	0.63	22 877.7	2963.02

Table E.31: Results of the different local search tests on environment tf3_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.1						
Sim.	0.01	0.01					
Sim.L			0.01				
Sim.I			0.01				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.		0.01					
Sim.L		0.01					
Sim.I		0.01					
Sim.T		0.01					
Sim.T.L		0.01					
Sim.T.I		0.01					

(b)

Table E.32: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.17 uulib

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	8.5	0.53	4.24	6.05	2197.2	3034.21	23.68	7.11	13197.2	3034.21
Tabu 10	9.	0.	0.39	0.02	296.	6.09	58.86	6.32	10296.	6.09
Sim.	9.6	0.84	1.05	0.51	6671.9	3347.9	2.27	0.3	16671.9	3347.9
Sim. L	9.	0.	0.04	0.	317.4	8.98	1.38	0.11	10317.4	8.98
Sim. I	9.	0.	0.03	0.01	324.9	13.34	1.41	0.07	10324.9	13.34
Sim. Tmin	8.7	0.48	0.78	0.55	4345.1	3195.23	2.1	0.32	14345.1	3195.23
Sim. Tmin L	8.9	0.32	0.04	0.	323.	8.58	1.34	0.08	10323.	8.58
Sim. Tmin I	8.9	0.32	0.04	0.01	321.9	7.59	1.3	0.09	10321.9	7.59

Table E.33: Results of the different local search tests on environment uulib. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.05					
Sim.L			0.05				
Sim.I			0.05				
Sim.T			0.01				
Sim.T.L			0.01				
Sim.T.I			0.01				

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.01						
Sim.	0.05						
Sim.L	0.01						
Sim.I	0.01						
Sim.T	0.02						
Sim.T.L	0.01						
Sim.T.I	0.01						

(b)

Table E.34: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.18 uulib_min

	|T|	**(dev)**	**TTB**	**(dev)**	**ITB**	**(dev)**	**TTE**	**(dev)**	**ITE**	**(dev)**
Tabu 5	9.2	0.42	2.18	3.73	1476.7	2402.4	15.	8.	12 476.7	2402.4
Tabu 10	9.	0.	0.29	0.02	249.1	3.14	45.12	5.56	10 249.1	3.14
Sim.	10.2	0.79	0.59	0.53	6119.6	5873.48	1.43	0.32	16 119.6	5873.48
Sim. L	9.6	0.7	0.02	0.01	281.	16.13	1.05	0.08	10 281.	16.13
Sim. I	9.8	0.42	0.02	0.	274.5	12.01	1.1	0.06	10 274.5	12.01
Sim. Tmin	9.5	0.53	0.57	0.34	4903.1	3722.12	1.38	0.25	14 903.1	3722.12
Sim. Tmin L	9.9	0.32	0.03	0.	280.9	12.82	1.08	0.09	10 280.9	12.82
Sim. Tmin I	9.7	0.48	0.02	0.	281.	14.74	1.04	0.1	10 281.	14.74

Table E.35: Results of the different local search tests on environment uulib_min. $|\mathbf{T}|$ is the number of transfers for the best solution, **TTB** is the time in seconds it took to find this solution. **ITB**, **TTE** and **ITE** are the iterations needed to reach the best solution, the total runtime and the total number of iterations.

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10							
Sim.	0.01	0.01					
Sim.L							
Sim.I		0.02					
Sim.T			0.1				
Sim.T.L	0.1	0.01					
Sim.T.I		0.1					

(a)

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L
Ta. 10	0.05						
Sim.							
Sim.L	0.02						
Sim.I	0.02						
Sim.T							
Sim.T.L	0.02						
Sim.T.I	0.02						

(b)

Table E.36: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different versions of local search. In Table (a) the scores of the best solutions are compared and in Table (b) the differences in runtime are compared.

E.19 Reason for termination

	Ta. 5	Ta. 10	Sim.	Sim.L	Sim.I	Sim.T	Sim.T.L	Sim.T.I
oil. min.	0.	0.	0.	0.	0.	0.	0.	0.
oil. or.	0.	0.	0.	0.	0.	0.	0.	0.
oil. sc. min.	0.	0.	0.	0.	0.	0.	0.	0.
oil. sc. or.	0.	0.	0.	0.	0.	0.	0.	0.
vert. min.	0.	0.	0.	0.	0.	0.	0.	0.
vert. or.	0.	0.	0.	0.	0.	0.	0.	0.
vert. sc. min.	0.	0.	0.	0.	0.	0.	0.	0.
vert. sc. or.	0.	**1.**	0.	0.	0.	0.	0.	0.
max min.	**0.2**	**0.9**	0.	0.	0.	0.	0.	0.
max or.	**0.2**	**1.**	0.	0.	0.	0.	0.	0.
tf1 min.	0.	0.	0.	0.	0.	0.	0.	0.
tf1 or.	0.	**0.3**	0.	0.	0.	0.	0.	0.
tf2 min.	**0.9**	**1.**	0.	0.	0.	0.	0.	0.
tf2 or.	**1.**	**1.**	0.	0.	0.	**0.2**	0.	0.
tf3 min.	0.	0.	0.	0.	0.	0.	0.	0.
tf3 or.	0.	**0.1**	0.	0.	0.	0.	0.	0.
uulib min.	0.	0.	0.	0.	0.	0.	0.	0.
uulib or.	0.	0.	0.	0.	0.	0.	0.	0.

Table E.37: This table lists the fraction of local searches that did not finish within 1800 seconds (one half hour) for each individual environment.

Appendix F

Saaltink's results

	Flood				Cut			
	|T|	(dev)	t (s)	(dev)	|T|	(dev)	t (s)	(dev)
as_oilrig	59.	0.	0.	0.	21.	0.	5.21	0.01
as_oilrig_scaled	240.	0.	0.03	0.01	205.	0.	601.81	0.29
de_vertigo	6.	0.	0.	0.	6.	0.	0.06	0.01
de_vertigo_scaled	133.	0.	0.01	0.01	109.	0.	16.35	0.01
max	195.	0.	1.04	0.01	---	---	---	---
tf1	1877.	0.	2.05	0.01	---	---	---	---
tf2	1607.	0.	1.07	0.01	---	---	---	---
tf3	832.	0.	0.23	0.01	---	---	---	---
uulib	31.	0.	0.	0.	13.	0.	1.34	0.01

Table F.1: The results found for the original environments using Saaltink's Flood-Fill and flow-based approach. For the environments max, tf1, tf2 and tf3 no results were obtained for the flow-based approach. The reason is that running the algorithm a single time lasted very long.

Appendix G

ILP results

G.1 LS Pricer Results

G.1.1 Score

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Score	**(dev)**	**Score**	**(dev)**	**Score**	**(dev)**	**Score**	**(dev)**
Tabu 1 u1	-177.759	147.925	-228.101	228.809	-49.125	35.91	-43.802	36.46
Tabu 2 u1	-347.568	101.488	-448.319	128.154	-69.629	32.235	-76.738	39.75
Sim. u1	-457.312	7.619	-584.697	6.83	-124.424	7.074	-144.29	6.2
Tabu 1 u2	-198.077	206.305	-194.852	400.876	-60.305	49.48	-61.675	64.715
Tabu 2 u2	-427.208	115.416	-543.65	161.012	-86.968	38.165	-99.988	46.394
Sim. u2	-559.821	13.748	-725.219	18.104	-149.354	10.116	-176.691	10.097
Tabu 1 u3	-254.055	275.13	-218.007	423.528	-63.122	60.28	-60.591	61.954
Tabu 2 u3	-507.082	151.021	-619.528	197.91	-109.173	47.046	-127.467	48.213
Sim. u3	-663.437	9.155	-859.667	16.737	-184.343	9.269	-216.958	12.011

Table G.1: The average scores for the different LS Pricer experiments. The value under the columns **Score** is calculated by subtracting the optimal score (found using the ILP Pricer) from the score found by the LS.

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Tabu 1	**Tabu 2**	**Tabu 1**	**Tabu 2**	**Tabu 1**	**Tabu 2**	**Tabu 1**	**Tabu 2**
Tabu 2	0.01		0.01		0.01		0.01	
Sim.	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

(a)

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Tabu 1	**Tabu 2**	**Tabu 1**	**Tabu 2**	**Tabu 1**	**Tabu 2**	**Tabu 1**	**Tabu 2**
Tabu 2	0.01		0.01		0.01		0.01	
Sim.	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

(b)

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Tabu 1	**Tabu 2**	**Tabu 1**	**Tabu 2**	**Tabu 1**	**Tabu 2**	**Tabu 1**	**Tabu 2**
Tabu 2	0.01		0.01		0.01		0.01	
Sim.	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

(c)

Table G.2: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different scores of the LS Pricers. Table (a) through (c) describe the results for $u = 1$ through $u = 3$.

G.1.2 Time

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Runtime	(dev)	Runtime	(dev)	Runtime	(dev)	Runtime	(dev)
Tabu 1 u1	0.145	0.066	0.148	0.076	0.093	0.031	0.151	0.046
Tabu 2 u1	0.037	0.018	0.045	0.028	0.044	0.018	0.057	0.029
Sim. u1	0.01	0.006	0.009	0.005	0.009	0.007	0.01	0.006
Tabu 1 u2	0.139	0.066	0.168	0.073	0.095	0.026	0.145	0.045
Tabu 2 u2	0.039	0.021	0.034	0.018	0.041	0.018	0.051	0.028
Sim. u2	0.009	0.005	0.009	0.007	0.01	0.007	0.011	0.006
Tabu 1 u3	0.117	0.071	0.178	0.071	0.092	0.031	0.141	0.046
Tabu 2 u3	0.041	0.02	0.042	0.022	0.038	0.016	0.054	0.026
Sim. u3	0.01	0.007	0.009	0.006	0.01	0.007	0.01	0.005

Table G.3: The average time used for each LS Pricer experiments. The time is measured in seconds.

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Tabu 1	Tabu 2	Tabu 1	Tabu 2	Tabu 1	Tabu 2	Tabu 1	Tabu 2
Tabu 2	0.01		0.01		0.01		0.01	
Sim.	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

(a)

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Tabu 1	Tabu 2	Tabu 1	Tabu 2	Tabu 1	Tabu 2	Tabu 1	Tabu 2
Tabu 2	0.01		0.01		0.01		0.01	
Sim.	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01

(b)

	as_oilrig		as_oilrig_min		uulib		uulib_min	
	Tabu 1	Tabu 2	Tabu 1	Tabu 2	Tabu 1	Tabu 2	Tabu 1	Tabu 2
Tabu 2	0.01		0.01		0.01		0.01	
Sim.	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

(c)

Table G.4: Results for the Anova significance test using Tukey as post-hoc analysis, comparing the different runtimes of the LS Pricers. Table (a) through (c) describe the results for $u = 1$ through $u = 3$.

G.2 Branch-and-Price results

	#LSP	cLSP	t LSP	#LPP	cLPP	tLPP	kick	branch	coll
as_oilrig_min	771	3204	683.81	277	5605	30 924.	15	15	15
as_oilrig_orig	684	3933	698.54	211	4150	30 643.7	15	15	15
as_oilrig_scaled_min	498	6281	828.12	76	7193	28 710.6	40	37	37
as_oilrig_scaled_orig	403	7802	854.67	28	2421	29 429.1	43	43	42
de_vertigo_min	145	1585	217.69	27	61	1098.72	2	2	2
de_vertigo_orig	171	2258	281.71	26	122	4385.53	2	2	2
de_vertigo_scaled_min	980	4701	907.62	314	13 789	29 663.8	5	5	5
de_vertigo_scaled_orig	530	6791	866.89	80	6314	28 697.	2	2	2
max_min	363	19 321	1505.48	362	362	0.36	74	74	74
max_orig	319	18 558	1508.1	319	319	0.32	72	72	72
tf1_min	2410	35 562	4718.16	2206	2206	2.21	689	511	498
tf1_orig	1596	36 785	4173.74	1521	1521	1.52	739	630	621
tf2_min	269	16 395	1139.74	270	270	0.27	722	722	722
tf2_orig	272	17 823	1166.39	273	273	0.27	707	707	707
tf3_min	1967	22 680	3178.57	1719	1719	1.72	371	216	214
tf3_orig	1729	24 738	3178.69	1	100	47 697.9	384	274	267
uulib_min	2361	1975	1261.03	1643	86 540	26 466.5	9	9	8
uulib_orig	1481	2355	919.13	782	30 077	64 402.6	9	9	8

Table G.5: Results for the ILP experiments using the LS Pricer. Columns #**LSP**, **cLSP** and **tLSP** show the number of executions, columns generated and time spent in the LS Pricer, whereas the columns #**LPP**, **cLPP** and **tLPP** show the number of executions, columns generated and time spent in the ILP Pricer. Under the columns **kick**, **branch** and **coll** the different scores obtained after the different phases are shown.

	#LSP	cLSP	t LSP	#LPP	cLPP	tLPP	kick	branch	coll
as_oilrig_min	767	3011	665.82	293	6320	30 461.6	18	18	18
as_oilrig_orig	790	4116	793.17	220	4400	30 244.9	18	18	18
as_oilrig_scaled_min	587	6397	896.9	86	7982	29 285.7	76	30	30
as_oilrig_scaled_orig	377	7400	799.03	27	2548	29 239.1	75	29	29
de_vertigo_min	120	1536	195.3	21	55	924.75	6	2	2
de_vertigo_orig	220	2376	327.5	26	123	3936.71	6	2	2
de_vertigo_scaled_min	39	2121	153.19	39	39	0.04	14	2	2
de_vertigo_scaled_orig	588	6901	917.03	78	6661	28 312.	12	2	2
max_min	371	19 422	1523.74	372	372	0.37	81	28	28
max_orig	354	19 589	1621.28	353	353	0.35	107	84	28
tf1_min	349	19 168	1474.53	349	349	0.35	878	78	78
tf1_orig	1094	32 404	3388.79	1070	1070	1.07	962	343	79
tf2_min	262	16 122	1147.31	263	263	0.26	1001	347	347
tf2_orig	278	17 881	1170.09	279	279	0.28	1006	337	337
tf3_min	1990	22 319	3175.6	1735	1735	1.74	559	156	156
tf3_orig	1828	25 831	3290.56	1635	1635	1.64	558	154	154
uulib_min	2453	2018	1307.3	1686	82 377	27 800.4	10	9	9
uulib_orig	1863	2546	1134.53	1086	33 987	27 249.7	9	9	9

Table G.6: Results for the ILP experiments using the LS Pricer in combination with the HH algorithm. Columns #**LSP**, **cLSP** and **tLSP** show the number of executions, columns generated and time spent in the LS Pricer, whereas the columns #**LPP**, **cLPP** and **tLPP** show the number of executions, columns generated and time spent in the ILP Pricer. Under the columns **kick**, **branch** and **coll** the different scores obtained after the different phases are shown.

Appendix H

TF results

	|T|	(dev)	|V|	(dev)	|E|	(dev)	|O|	(dev)	t (s)	(dev)
oil. min.	29.15	0.59	279.	0.	475.	0.	183.75	1.86	1.27	0.04
oil. or.	24.45	1.96	412.	0.	633.6	0.5	231.3	0.57	1.42	0.04
oil. sc. min.	88.45	1.82	287.2	1.44	411.5	1.76	164.05	4.7	9.21	0.29
oil. sc. or.	89.55	4.12	611.15	1.04	753.85	2.03	336.7	4.17	7.6	0.51
vert. min.	6.	0.	237.	0.	452.	0.	12.	0.	2.22	0.01
vert. or.	6.	0.	346.	0.	603.	0.	12.	0.	2.1	0.01
vert. sc. min.	12.	0.	450.	0.	683.	0.	24.	0.	3.61	0.01
vert. sc. or.	12.	0.	997.	0.	1234.	0.	16.	0.	4.44	0.04
max min.	80.	0.	112.35	0.75	130.2	1.11	879.8	13.34	156.35	3.16
max or.	80.	0.	67.6	0.6	80.4	1.19	623.15	11.15	253.5	9.91
tf1 min.	889.9	2.85	1363.85	3.96	1818.25	4.42	8290.75	56.48	228.23	21.13
tf1 or.	883.35	4.34	2080.9	4.48	2502.75	4.28	11 090.3	106.73	237.32	26.12
tf2 min.	1048.8	4.75	2876.	3.36	4551.95	5.83	1916.4	13.53	97.19	1.97
tf2 or.	1033.45	4.24	3213.55	3.99	4796.6	4.13	2567.15	22.76	142.04	3.74
tf3 min.	576.85	3.2	1071.3	2.66	1664.35	4.23	1047.8	9.89	43.67	2.22
tf3 or.	574.85	4.83	1377.7	1.81	1962.8	3.	1312.35	9.44	59.67	2.78
uulib min.	9.25	0.44	42.3	0.47	54.3	0.47	32.2	1.15	2.53	0.03
uulib or.	9.4	0.5	24.4	0.5	28.4	0.5	32.8	1.79	6.15	0.16

Table H.1: The results for the TF experiments. Column $|\mathbf{T}|$ shows the cuts found for the environments. The values listed under $|\mathbf{V}|$, $|\mathbf{E}|$ and $|\mathbf{O}|$ are the number of vertices, edges and overlaps in the resulting environments.

Appendix I

Difference between original and minimized environments

I.1 SPH

	Score	sig	time	sig
as_oilrig	4.7	0.01	0.028	
as_oilrig_scaled	4.9	0.01	6.953	0.01
de_vertigo	0.3	0.05	0.009	0.01
de_vertigo_scaled	-1.	0.01	0.214	0.01
max	5.55	0.01	239.149	0.01
tf1	-2.8		6723.64	0.01
tf2	54.3	0.01	32.205	0.01
tf3	5.7	0.05	99.646	0.01
uulib	3.4	0.01	0.021	0.01

Table I.1: Difference of $|\mathbf{T}|$ (listed under **Score**) and time between minimized and original environments for each environment. Positive values mean better results for the minimized versions of the environments. Under **sig** the significance level of the difference is listed.

I.2 HH

	Score	sig	time	sig
as_oilrig	-0.35	0.05	0.003	
as_oilrig_scaled	-1.9	0.01	0.023	0.01
de_vertigo	0.		0.004	0.02
de_vertigo_scaled	0.		0.005	0.01
max	0.		0.128	0.01
tf1	-5.6	0.01	0.359	0.01
tf2	-8.25	0.01	0.809	0.01
tf3	1.15	0.01	0.144	0.01
uulib	0.		-0.001	

Table I.2: Difference of $|\mathbf{T}|$ (listed under **Score**) and time between minimized and original environments for each environment. Positive values mean better results for the minimized versions of the environments. Under **sig** the significance level of the difference is listed.

I.3 Local Search

	as	as_s	de	de_s	max	tf1	tf2	tf3	uu
Tabu 5	-10.	-0.8	-9.	-1.6	-2.4	17.1	2.5	-16.8	-0.7
Tabu 10	-4.2	4.4	-4.9	-1.2	-2.4	53.1	10.2	6.9	0.
Sim.	-18.3	0.3	-28.3	-2.	-11.4	106.6	64.5	-16.2	-0.6
Sim. L	-17.6	1.2	-23.5	-3.	-1.9	38.7	0.5	-1.1	-0.6
Sim. I	-18.	-0.9	-20.2	-0.3	-2.	36.3	-5.5	-6.8	-0.8
Sim. Tmin	-21.1	-4.1	-27.6	-2.	-2.9	47.6	6.5	-5.3	-0.8
Sim. Tmin L	-17.8	3.2	-14.2	-1.7	-1.9	37.5	-11.8	-3.4	-1.
Sim. Tmin I	-17.9	0.1	-17.8	-1.4	-1.7	41.1	1.5	-4.2	-0.8

Table I.3: Difference of $|\mathbf{T}|$ between minimized and original environments for each environment. Positive values mean better results for the minimized versions of the environments.

	as	as_s	de	de_s	max	tf1	tf2	tf3	uu
Tabu 5					0.01	0.01	0.01	0.01	0.01
Tabu 10		0.01	0.01	0.02	0.02	0.01	0.01	0.01	0.01
Sim.	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02
Sim. L	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Sim. I					0.01	0.01	0.01	0.01	0.01
Sim. Tmin	0.01	0.05	0.05	0.05	0.01	0.01	0.01	0.01	0.01
Sim. Tmin L		0.02	0.02	0.05	0.05	0.01	0.02	0.02	0.01
Sim. Tmin I					0.01	0.01	0.01	0.01	0.01

Table I.4: Test for significance of the differences given in Table I.3

	as	as_s	de	de_s	max	tf1	tf2	tf3	uu
Tabu 5	3.13	126.7	0.55	31.77	-146.7	198.98	268.56	85.67	2.06
Tabu 10	1.49	117.48	1.26	146.	59.36	408.14	118.32	310.91	0.1
Sim.	1.81	5.99	-1.64	34.47	3.02	27.28	24.65	5.29	0.46
Sim. L	0.04	0.69	0.02	0.31	5.68	16.05	21.63	3.73	0.01
Sim. I	0.08	0.69	0.02	0.31	8.13	17.5	21.95	4.6	0.01
Sim. Tmin	7.81	72.54	-2.07	23.67	30.81	74.42	281.77	19.27	0.21
Sim. Tmin L	0.08	0.35	0.02	0.31	8.7	18.16	22.41	4.24	0.01
Sim. Tmin I	0.06	1.12	0.02	0.32	6.96	17.93	20.13	3.58	0.02

Table I.5: Difference of time needed to reach the best solution between minimized and original environments for each environment. Positive values mean faster results for the minimized versions of the environments.

	as	as_s	de	de_s	max	tf1	tf2	tf3	uu
Tabu 5	0.02	0.01	0.01	0.02	0.02	0.01	0.01	0.01	0.01
Tabu 10			0.01	0.01	0.01	0.02	0.02	0.02	0.01
Sim.	0.05	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Sim. L		0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Sim. I	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Sim. Tmin	0.01	0.01	0.01	0.01	0.01	0.01	0.05	0.01	0.01
Sim. Tmin L	0.05	0.05	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Sim. Tmin I	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Table I.6: Test for significance of the differences given in Table I.5

I.4 TF

	Score	sig	time	sig
as_oilrig	-4.7	0.01	0.148	0.01
as_oilrig_scaled	1.1		-1.61	0.01
de_vertigo	0.		-0.112	0.01
de_vertigo_scaled	0.		0.833	0.01
max	0.		97.151	0.01
tf1	-6.55	0.01	9.089	
tf2	-15.35	0.01	44.854	0.01
tf3	-2.		16.006	0.01
uulib	0.15		3.62	0.01

Table I.7: Difference of $|\mathbf{T}|$ (listed under **Score**) and time between minimized and original environments for each environment. Positive values mean better results for the minimized versions of the environments. Under **sig** the significance level of the difference is listed.

Appendix J

Comparing results

J.1 as_oilrig

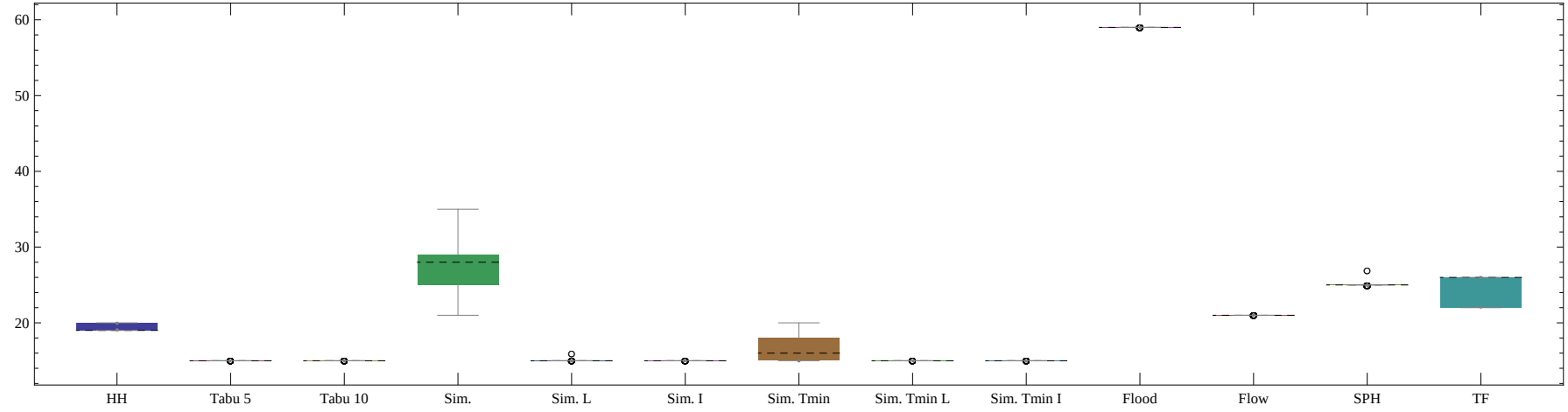


Table J.1

J.2 as_oilrig_min

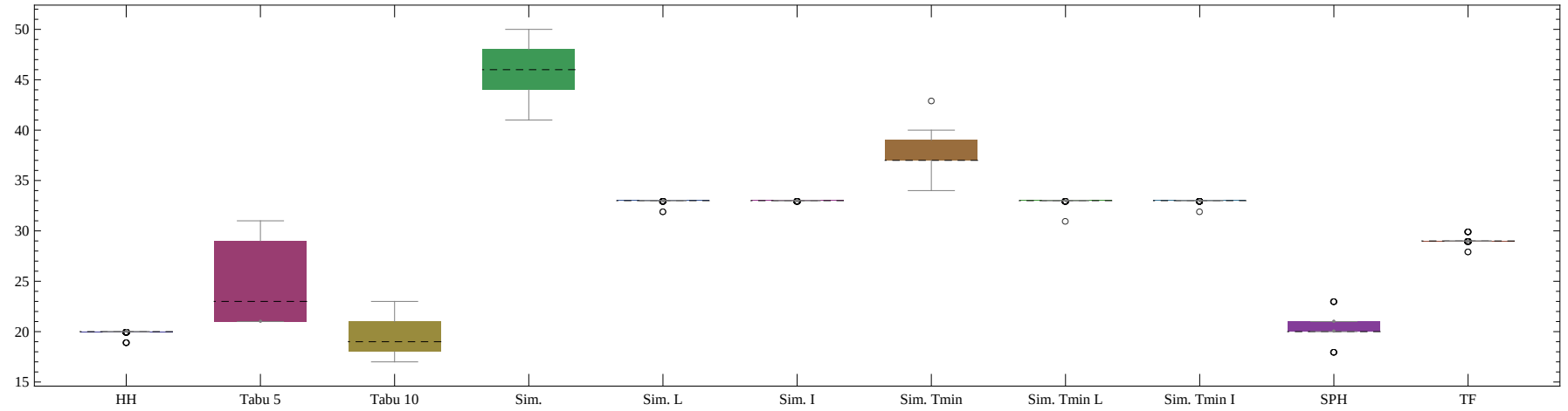


Table J.2

J.3 as_oilrig_scaled

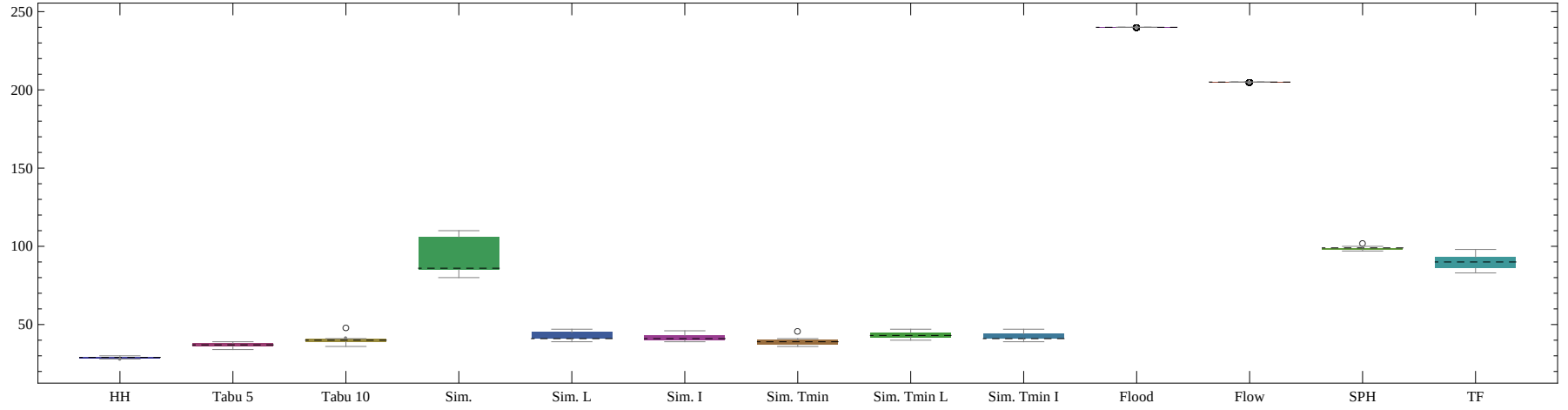


Table J.3

J.4 as_oilrig_scaled_min

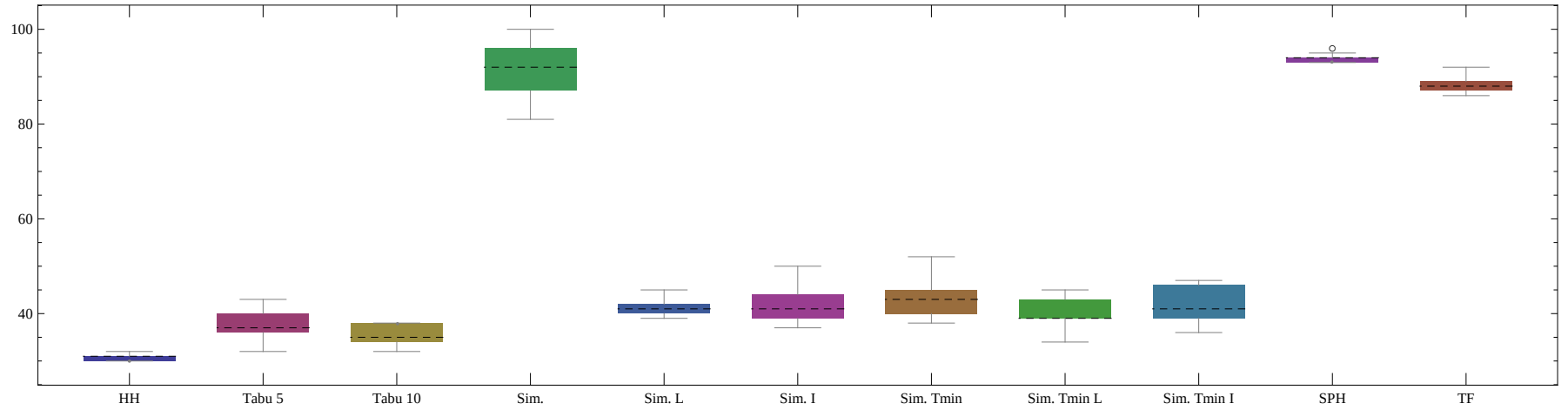


Table J.4

J.5 de_vertigo

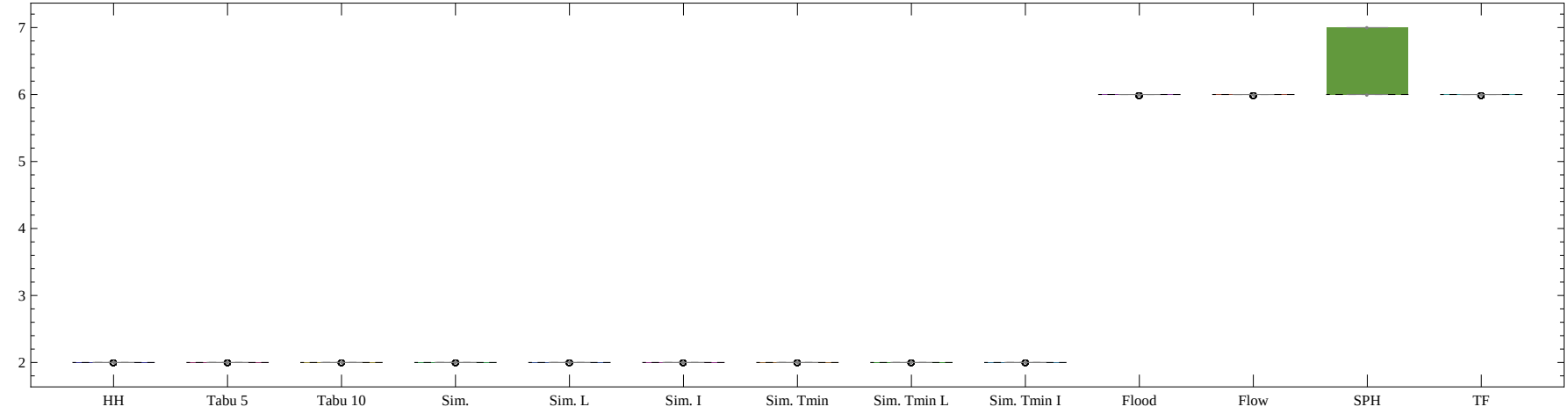


Table J.5

J.6 de_vertigo_min

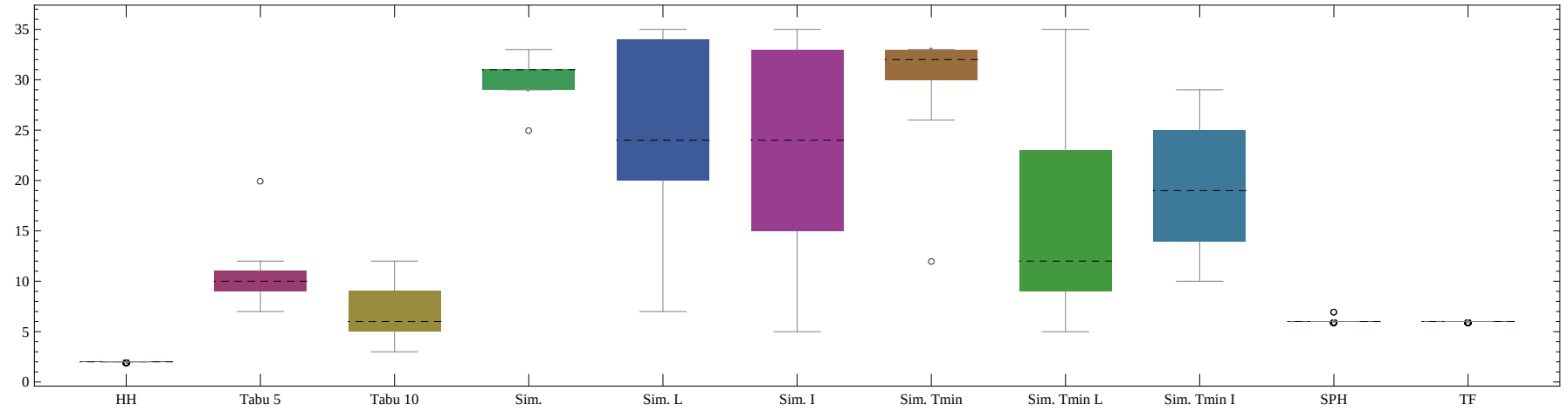


Table J.6

J.7 de_vertigo_scaled

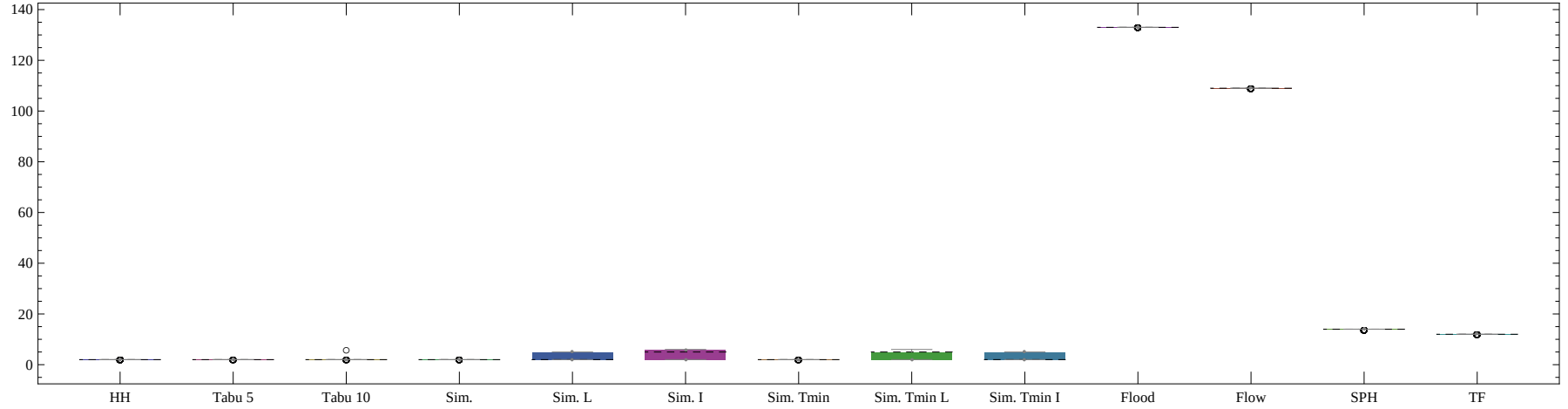


Table J.7

J.8 de_vertigo_scaled_min

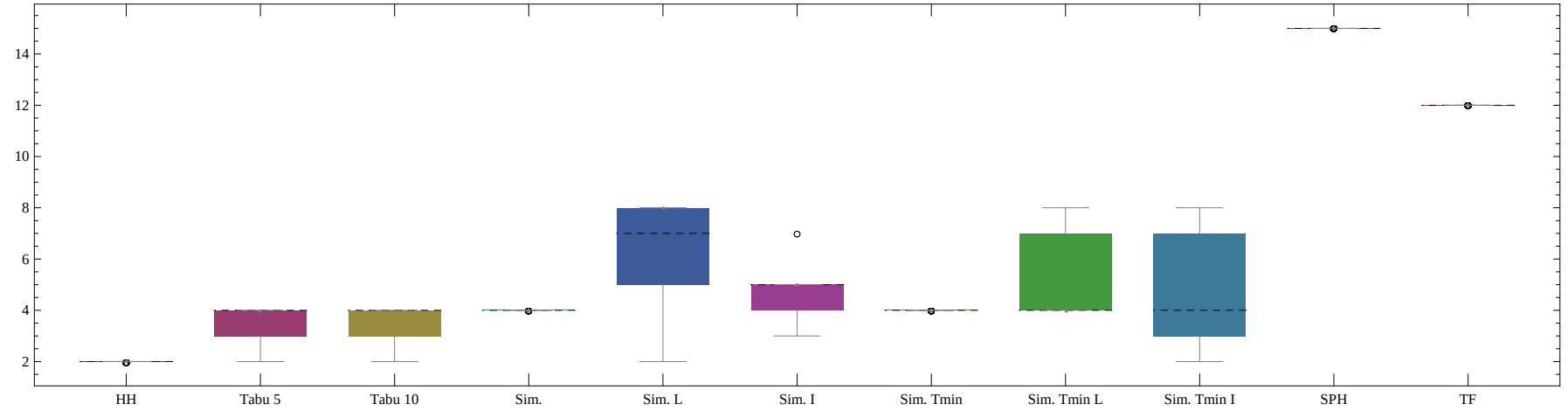


Table J.8

J.9 max

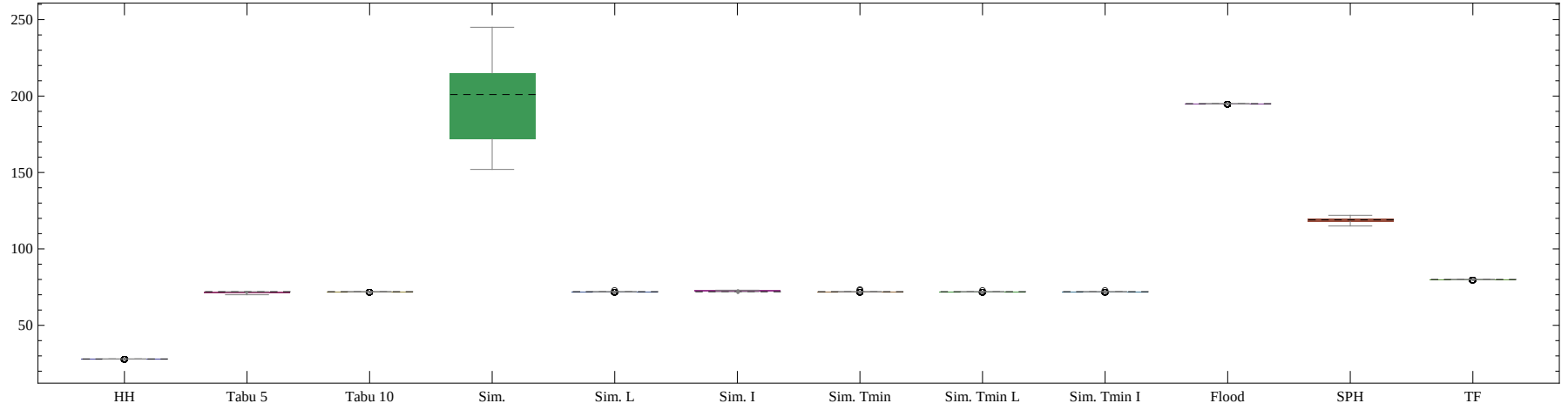


Table J.9

J.10 max_min

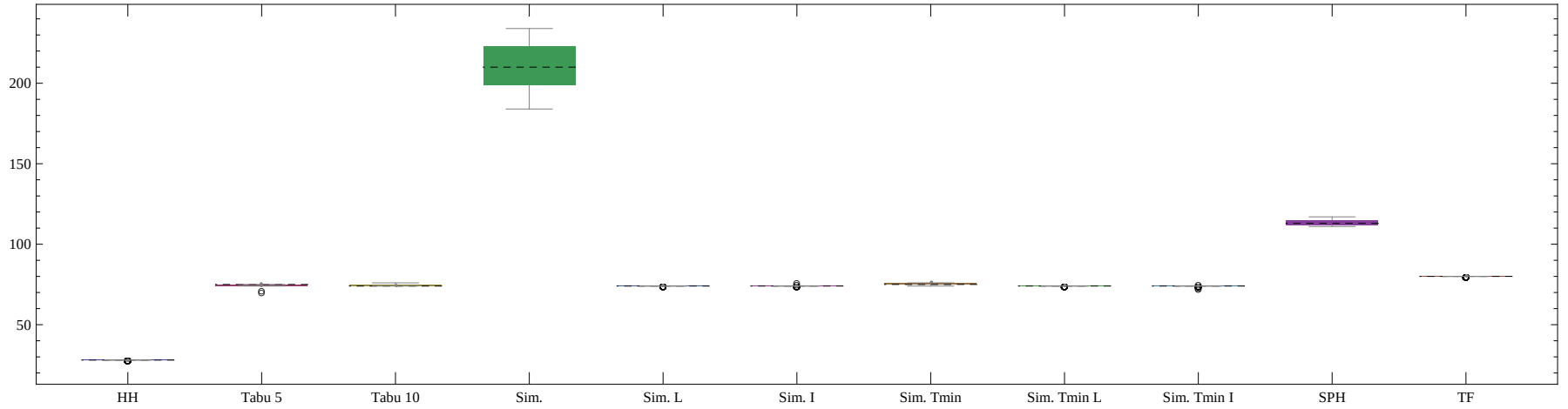


Table J.10

J.11 tf1

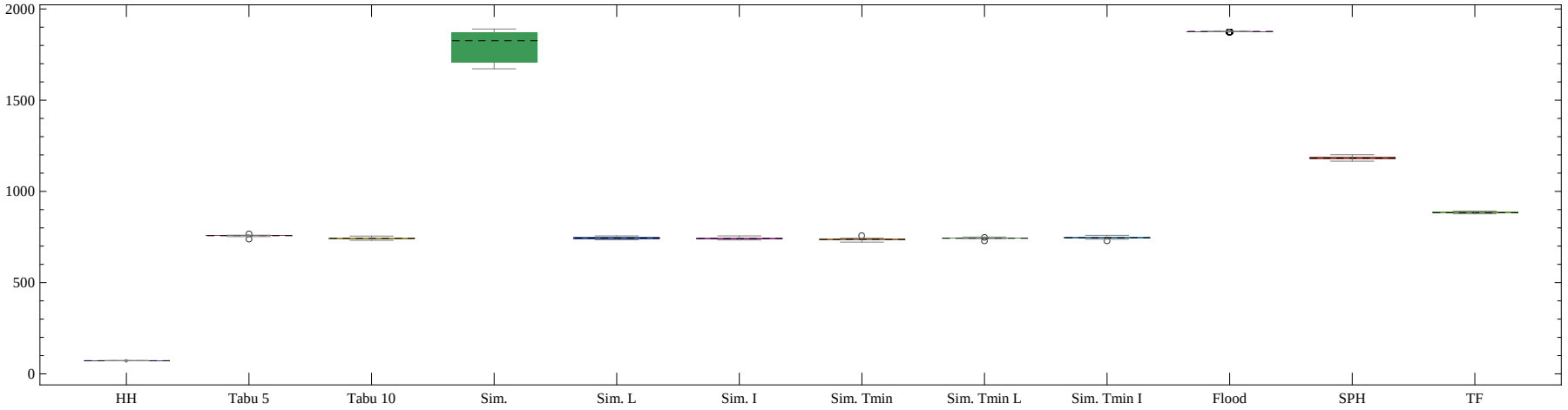


Table J.11

J.12 tf1_min

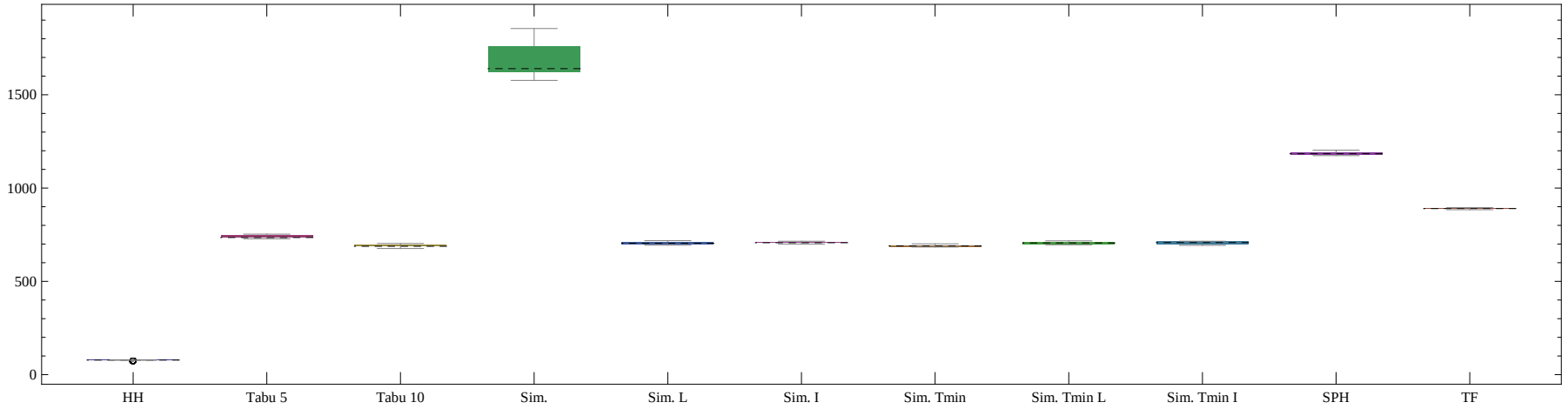


Table J.12

J.13 tf2

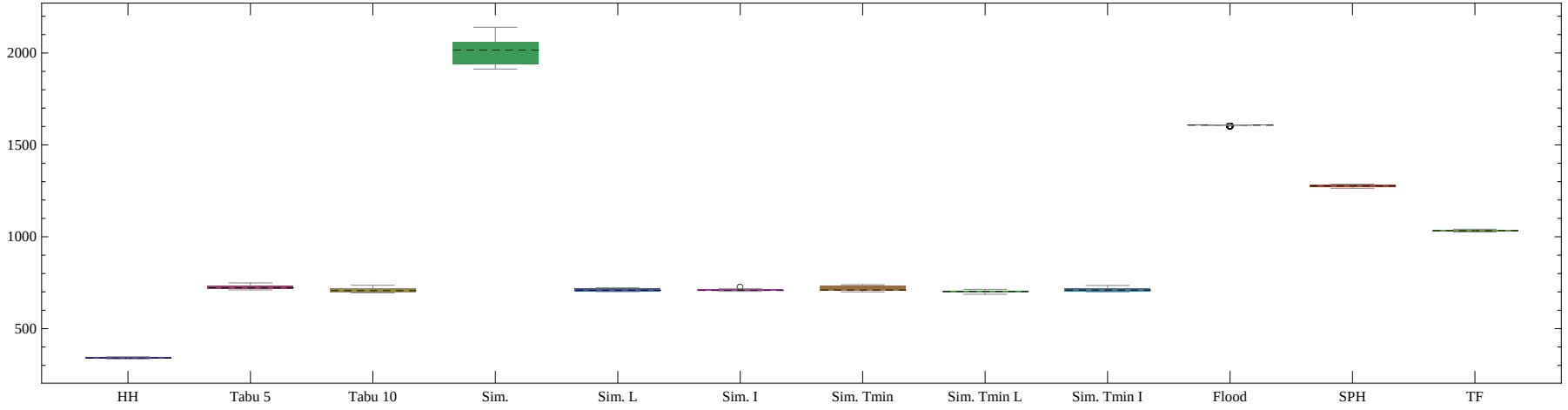


Table J.13

J.14 tf2_min

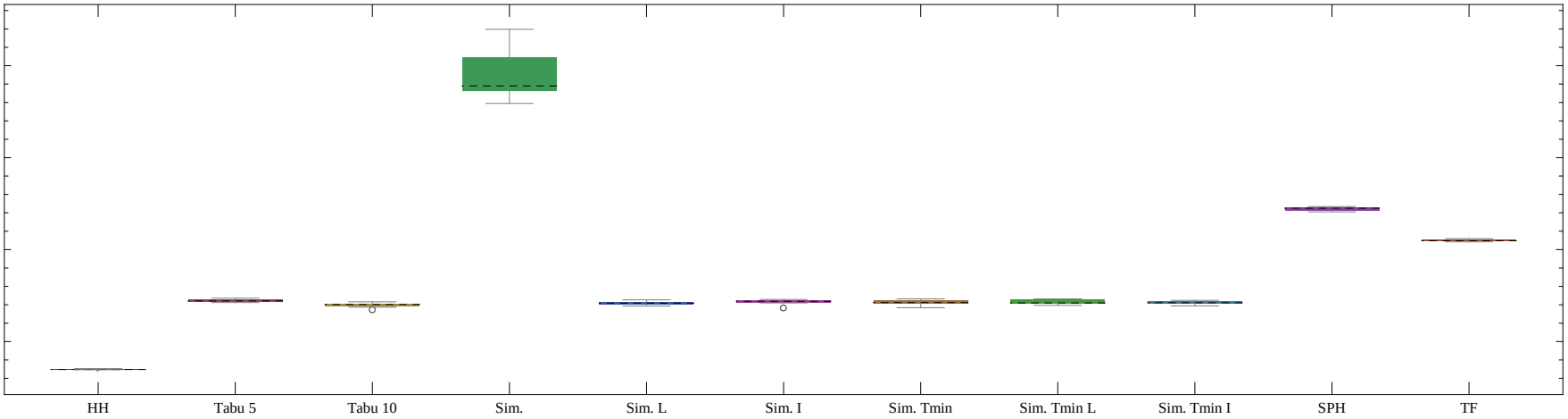


Table J.14

J.15 tf3

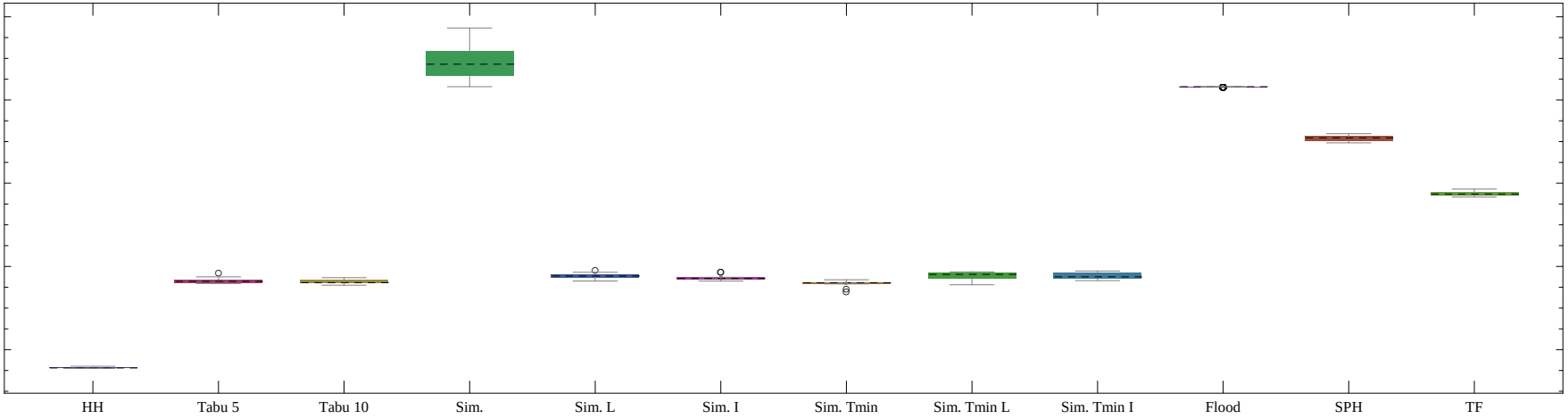


Table J.15

J.16 tf3_min

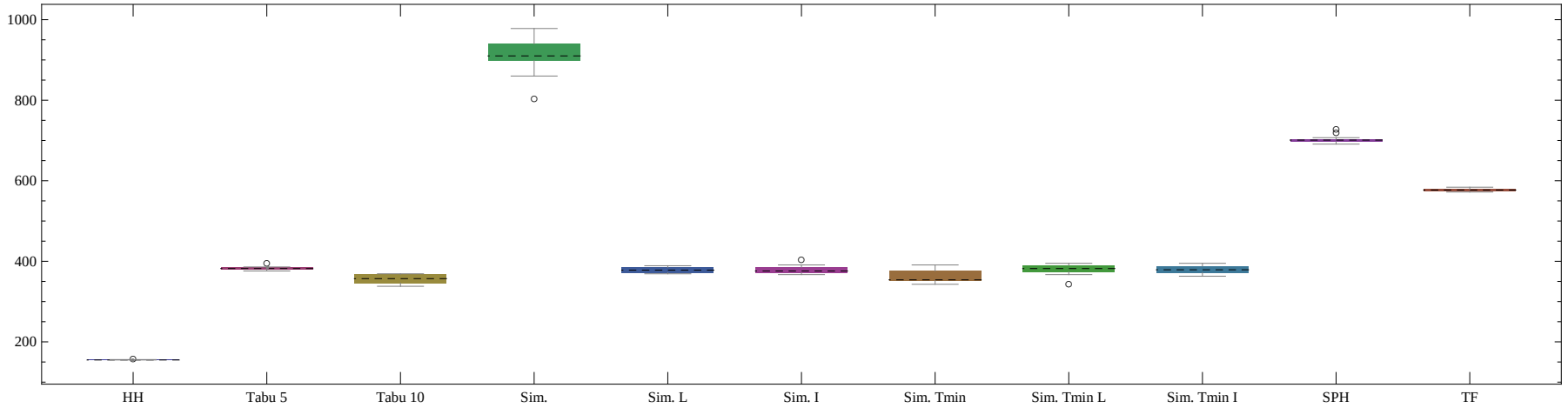


Table J.16

J.17 uulib

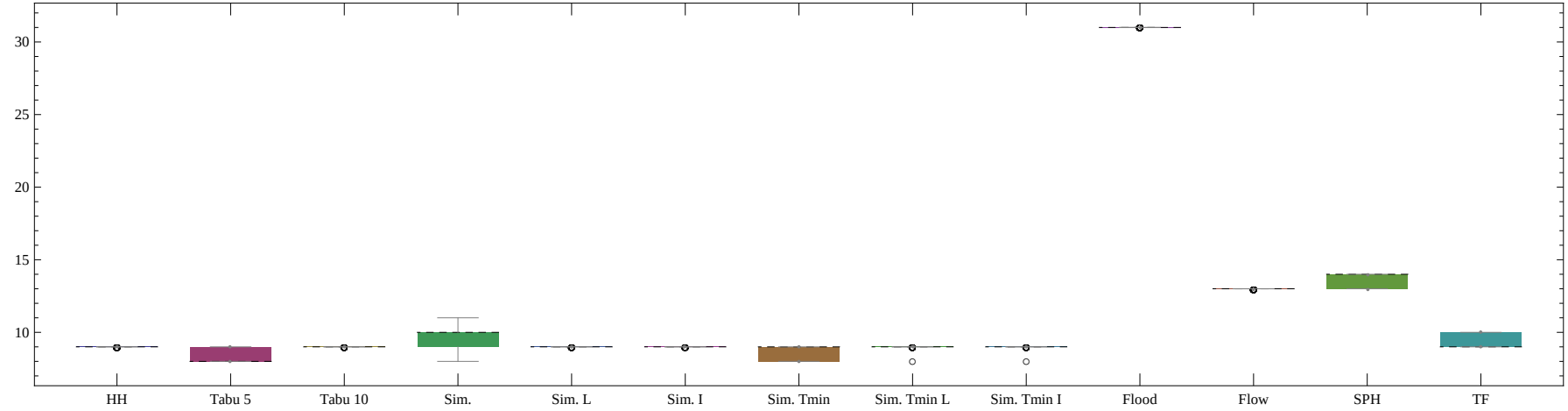


Table J.17

J.18 uulib_min

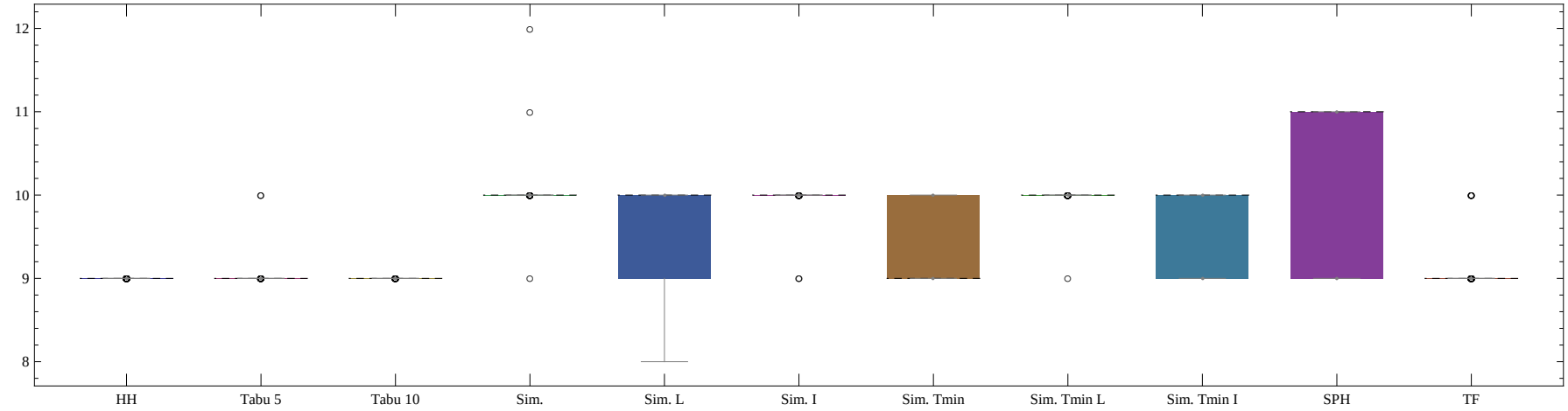


Table J.18