CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Linear Arrangements with Closeness Constraints

Master's thesis in Computer science and engineering

Pedram Shirmohammad

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Linear Arrangements with Closeness Constraints

Pedram Shirmohammad

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Linear Arrangements with Closeness Constraints

Pedram Shirmohammad

Supervisor: Peter Damaschke, Department of Computer Science and Engineering
Examiner: Patrik Jansson, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Linear Arrangements with Closeness Constraints
Pedram Shirmohammad
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In this thesis we will examine the product location problem on a pick-by-order policy. We restrict the scope of this thesis to a single rack with a front depot and each order can consist of at most two products. We present a polynomial solution for an arbitrary graph where the optimal arrangement of each component is given and present a polynomial solution for forests consisting of trees of, at most, order three.

Keywords: Computer science, linear arrangement, product location problem, order picking, graphs.

Acknowledgements

I would like to thank Peter Damaschke for all his support, guidance and quick feedback throughout this thesis. Without him this thesis would not have been possible. I would also like to thank Patrik Jansson, my examiner, for his valuable feedback.

Pedram Shirmohammad, Gothenburg, June 2020

Contents

List of Figures

1

Introduction

Linear arrangement problems with closeness constraints are a set of optimization problems that require a set of vertices, in a graph, to be linearly arranged such that a specific function on the arrangement is optimized. This function varies depending on the objective of the problem. A selection of these objectives are minimizing the total edge length, minimizing the diameter and minimizing the number of edges in the largest cut. One can read more about these problems in [4]. In this thesis we will focus on a relatively new problem definition within this area that is, for now, mainly applied to product location in warehouses.

One of the main functions of a warehouse is order picking. Order picking is both time consuming and labour intensive and it is therefore no surprise that an extensive amount of research has been conducted within this area. There are many factors that contribute to an efficient order picking system such as the layout of the storage, picking route and product location. As mentioned before, in this thesis we will focus on optimization of the product location. The product location is dependent on how the orders are retrieved and the layout of the warehouse. We will examine the product location problem on a pick-by-order policy, which means that only the products in one order are picked in a single route. Moreover, we restrict the problem to a single rack with a front depot, see Figure 1.1. In more detail, assume we have a set of orders O and each order $o \in O$ consists of some products. To retrieve the products for an order the worker has to walk from the depot to the farthest product, in the order, and return back to the depot. The objective is then to place the distinct products such that the traveling distance for the order set is minimized. This problem has been proven to be NP-hard in a strong sense even for orders consisting of at most two products [3].

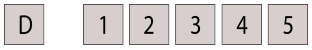


Figure 1.1: Single rack with depot marked as D. Each box with a number represents a storage on the rack.

To the best of our knowledge, there exists few studies that treat the product location problem on this layout. This might be due to the fact that the layout is simpler than layouts in many real-world warehouses that often consists of various numbers of shelves and dimensions. However, the problem could give a good insight to the complexity of other, more complicated, layouts within this field [9]. Moreover, exact

solutions to this layout could be applicable to the more complicated layouts since they often consist of the same basic structure.

Due to the lack of research in this problem we will investigate the problem on orders consisting of at most two products to understand the problem better. The following sections in this chapter formally defines the problem, the aim of the thesis and sets the limitations of this thesis.

1.1 Problem Definition

As previously mentioned, we will consider the restricted product location problem where each order consists of two distinct products. One could view each product as a vertex in a graph and each order as an edge incident to two vertices. The structure of a single rack with a front depot, see Figure 1.1, is similar to a linear arrangement. We define the linear arrangement of a graph $G = (V, E)$ as a permutation $\pi : V \to \{1, ..., n\}$ where $\pi(u)$ is the position of vertex u in the linear arrangement. Note that $\pi(u)$ can be viewed as the location of product u on a rack. Then, a formal definition of the problem can be described as follows.

For a graph $G = (V, E)$ let π_G be a linear arrangement of G. Following the description in the introduction, if a worker has to retrieve an order $(u, v) \in E$ then the worker has to travel $2 \cdot max\{\pi(u), \pi(v)\}$. Since 2 is just a constant in all orders, it does not contribute anything to the problem formulation and can be neglected. The objective is then to minimize the cost function

$$C(G, \pi_G) = \sum_{(u,v)\in E} max\{\pi_G(u), \pi_G(v)\} \tag{1.1}$$

Furthermore, a relevant extension is to add weight $w : E \to Q^+$ to each edge, since this can represent the priority or frequency of each order. Then the objective would be

$$C(G, \pi_G) = \sum_{(u,v)\in E} w(u, v) \cdot max\{\pi_G(u), \pi_G(v)\} \tag{1.2}$$

1.2 Aim of the Project

The aim of this project is to conduct a literature review on related problems and to explore the problem described in the previous section for special graph classes. More precisely, we will explore the problem for sparse graphs, weighted sparse graphs and an objective will is to develop an efficient algorithm for weighted sparse graphs.

Since the problem is proven to be NP-hard [3] by exploring the problem for sparse graphs we aim to obtain better results than in the general case.

1.3 Limitations

In this thesis we do not consider arrangements in two or more dimensions due to the increased complexity of the problem and the limited time frame of the project. Moreover, we only consider undirected graphs due to the formulation of the problem.

2

Literature Review

To the best of our knowledge, there exist few papers that examine this problem, and most of these papers analyze the problem from a heuristic point of view. Having said that, we will go through some of these studies that have contributed most to the problem in Section 2.1. Also, due to the lack of studies on this problem we will discuss the Bandwidth problem in Section 2.2, since it has some similarities to our problem.

2.1 Product Location Problem

The product location problem is defined as: Given a set of orders O, where each order $o \in O$ consists of some products, arrange the products on racks such that the traveling distance is minimized for the order set. As mentioned before, the layout and positioning of the racks can vary. We will examine papers that treat the problem for a single rack with a front depot, similar to Figure 1.1, since this layout is also the layout we treat in this thesis.

According to [3], one of the earliest studies that treats this problem is the paper published by van Oudheusden and Zhu [9]. The product location on a single rack is mentioned briefly, in [9], and an algorithm is presented for the restricted version where all orders are disjoint. Moreover, it is mentioned that the problem would be interesting to examine from an analytical point of view, since the layout with a single rack is somewhat easier than other layouts and thus could give good insights into the complexity of other, more complex, layouts.

Boysen and Stephan's paper [3] is one of the few papers that mainly focus on the product location problem. As mentioned in the introduction, they prove that the restricted problem to only two products per order is NP-hard in the strong sense even if each order consists of at most two products. This is proven by a reduction from the Optimal Linear Arrangement problem on graphs to the product location problem on a single rack. Furthermore, a dynamic programming (DP) solution is proposed, that is very similar to the DP scheme for sequencing problems proposed by Held and Karp [5]. The general idea of this algorithm is, starting with only one storage location, to try all possible arrangements and at each iteration add one more storage location and repeat until there are as many storage locations as products. The algorithm has a time complexity of $O(m \cdot n^2 \cdot 2^n)$, where n is the number of products and m is the number of orders.

Moreover, they propose a heuristic solution, named $G2$ that has achieved some good results, and this is showcased in both [3] and in [2]. In $G2$, each product is greedily arranged according to the number of occurrences in the order set, such that the product with most number of occurrences is placed at the first storage. In Section 3.3.1 we will apply the $G2$ algorithm to find the optimal arrangement for a weighted tree of, at most, order three.

2.2 Bandwidth on Caterpillars

The bandwidth problem (BP) is another linear arrangement problem defined as: Given a graph $G = (V, E)$, find a linear arrangement π_G of G such that $b(G, \pi_G) = max_{(u,v)\in E}|\pi_G(u) - \pi_G(v)|$ is minimized. The problem was proven to be NP-hard by Papadimitriou in [8] and it even remains NP-hard for various special graphs, such as caterpillars with hair-length at most three and caterpillars with at most one hair per backbone [7]. (We will give a formal definition of caterpillars later in this section.) We examine two solutions of the problem on restricted caterpillars, that are similar to each other. The reason why we specifically examine these solutions is because they are some of the few solutions that solve the BP for weighted graphs.

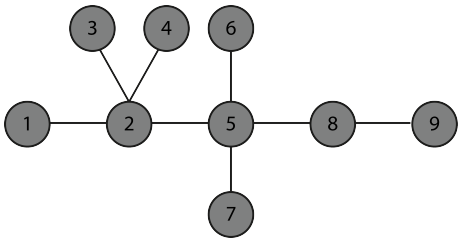
A caterpillar $C = (V, E)$ is a tree that consists of a simple path called backbone and line graphs connected to the backbone called hair, see Figure 2.1. Furthermore, let $D(C)$ be the diameter of C, that is the longest shortest path between any pair of vertices in C. Note that $D(C)$ is a path and, unlike the common conception of a diameter, not a number. The decision version of BP asks if there exists an arrangement of a graph, lets say C, such that $b(C, \pi_C) = m$, where m is a positive integer. In 1981, Assman et al. [1] introduced a polynomial solution for the BP on caterpillars with hair length at most two. They defined an algorithm for the decision version of the BP that either finds an arrangement such that $b(C, \pi_C) = m$ or $b(C, \pi_C) > m$. With this algorithm they also solve the BP.

The algorithm starts by assigning positions, for all l vertices on $D(C)$, along the path, accordingly $0, m, 2m, ..., dm$, i.e positioning them such that there is m distance between each vertex on $D(C)$. Then, for $k = 0, ..., l$, hair vertex u_1, adjacent to backbone vertex v, where $\pi_C(v) = k \cdot m$, is positioned within m distance from v. The second vertex on a hair is positioned within m distance to its adjacent hair vertex u_1. Thus, if a hair vertex is not able to be positioned within this distance then $b(C, \pi_C) > m$.

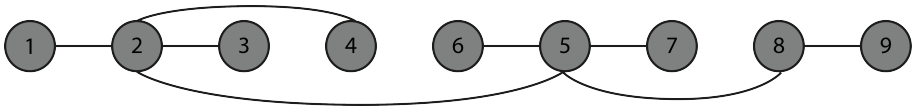
Naturally, $b(C, \pi_C)$ for a caterpillar C has to be between 1 and n. Thus, one can find the minimal $b(C, \pi_C)$ by searching through the span of 1 to n for the minimal m where $b(C, \pi_C) = m$, with the help of the described algorithm [1].

A similar approach has been used in [6] that treats the BP for weighted caterpillars with hair length of at most 1. The algorithm presented in [6] runs in $O(n \log n \log(nw_{max}))$ time where n is the number of vertices and w_{max} is the maximum edge weight. Similarly to the algorithm described above, they first define an algorithm for the decision version of the problem and then feed the algorithm with various values for m.

Now, let $C' = (V', E')$ be a caterpillar with hair length of at most one and this time each edge $(u, v) \in E'$ has a weight $w(u, v)$, that is a positive integer. Also, let the distance of an edge (u, v) be $d(u, v) = \frac{1}{w(u,v)}$. Given an integer m their algorithm checks if there exists an arrangement such that the $b(C', \pi_{C'}) = m$. Note that since each edge has a weight $b(C', \pi_{C'}) = max_{(u,v)\in E'} w(u, v) \cdot |\pi_{C'}(u) - \pi_{C'}(v)|$. The algorithm positions each vertex in $D(C')$ as far as possible from its adjacent vertices in $D(C')$ without any of the vertices in $D(C')$ violating $w(u, v) \cdot |\pi_{C'}(u) - \pi_{C'}(v)| < m$. Then, each hair vertex v is positioned such that it is within $m \cdot d(v, u)$ distance to all its adjacent vertices u. If this is not possible then $b(C', \pi_{C'}) > m$.



(a) A caterpillar with hair length at most 2.



(b) A linear arrangement of the caterpillar in (a), with $b(C, \pi_C) = 4$ as witnessed by the length 4 of the edge between nodes 2 and 5.

Figure 2.1: A caterpillar and its arrangement.

3

Results

In Section 3.1 we present some definitions and concepts and in Section 3.2 we present a polynomial solution for arbitrary graphs given the optimal arrangement of each component. Initially, in Section 3.2, we solve the problem for graphs without self-loops, i.e. graphs where there exists no edges that has both ends at the same vertex.

3.1 Preliminaries

3.1.1 Graphs

Let $H = (V, E)$ be a graph consisting of k components G_i, and let each edge $(u, v) \in E$ have an associated weight $w(u, v)$ where $w : E \to Q^+$. Let $d(v)$ denote the degree of $v \in V$. We denote an optimal arrangement of H as π^*_H and an optimal, independent, arrangement of each component G_i is denoted as $\pi^*_{G_i}$.

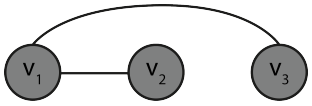
3.1.2 Arrangement

A linear arrangement π can be viewed as a horizontal line of vertices. For a position $\pi(v)$, where v is an arbitrary vertex in an arrangement π, we will sometimes refer to a vertex u where $\pi(u) < \pi(v)$, as a vertex left of v. Similarly, if $\pi(u) > \pi(v)$ we will refer to it as a vertex right of v. Also, since all arrangements, except for arrangements in general cases, will have a subscript of which graph/component the arrangement is for we reduce the notation of the cost function in Section 1.1 from $C(G, \pi_G)$ to $C(\pi_G)$.

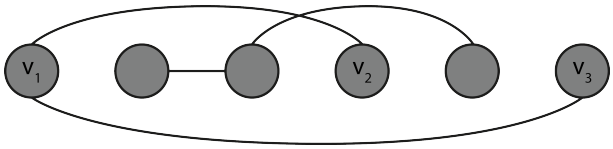
We introduce the notion of structure for a set of vertices in an arrangement as: For an arrangement π_G of a graph $G = (V, E)$, where $|V| = n$, the structure of π_G is the relative position of the vertices in π_G, i.e. where each vertex is positioned in relative to all other vertices in π_G. More formally, the structure of π_G is, for each $v_j \in V$, whether $\pi_G(v_j) < \pi_G(v_i)$ or $\pi_G(v_j) > \pi_G(v_i)$ for $i = 1, ..., n$ and $i \neq j$. For instance, assume G is a component in the graph H and that there exists at least one more component in H. In Figure 3.1, (a) is an arrangement π_G of the component G and (b) is an arrangement π_H of H. The structure of G in π_G is such that v_3 is positioned to the right of v_2 and v_1, v_2 is positioned between the other two vertices and finally v_1 is positioned to the left of both vertices. Moreover, we say that the structure of G in π_G is the same in π_H since:

- v_2 is still positioned between the other two vertices in G.
- v_3 is positioned to the right of the other two vertices in G
- v_1 is positioned to the left of the other two vertices in G

In other words, the relative positions of the vertices in G in π_G is the same as in π_H.



(a) Illustrates the arrangement π_G.



(b) Illustrates the the arrangement π_H.

Figure 3.1: The arrangement of π_G and π_H. The vertices of G have the same notation in both arrangements.

3.1.3 Chains and Subchains

In Algorithm 1 we will swap sets of consecutively positioned vertices to find an optimal arrangement. We define a set of consecutively positioned vertices of G_i in π_H as a chain. Let S_1 be an arbitrary chain of G_i in π_H and let v_l and v_r be the leftmost and rightmost vertices of S_1 in π_H. Note that there cannot exist a vertex that is not in S_1 and is positioned between v_l and v_r. In other words, the chain S_1 is the set of vertices positioned at $\pi(v_l), \pi(v_l)+1, \pi(v_l)+2, ..., \pi(v_l)+|S_1|-1$, where $|S_1|$ is the number of vertices in S_1. As can seen in Figure 3.2, there does not need to exist edges between the vertices in a chain. Furthermore, a chain's start and end vertex can be specified. A subchain S_2 of a chain S_1 is a subset of the vertices in S_1 such that the vertices in S_2 are consecutively positioned in π_H. It is possible for both chains and subchains to consist of a single vertex.

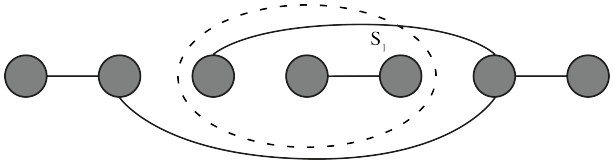


Figure 3.2: A linear arrangement with a chain S_1.

3.1.4 Cost Contribution

For a vertex v, take any of its adjacent vertices u. If $\pi(u) < \pi(v)$ then, following the definition of cost function 1.2 in Section 1.1, the weight of the edge (v, u) is multiplied with $\pi(v)$ in the cost function. Thus one could say that v contributes with $w(v, u)$ to the cost function, since $w(v, u)$ will be multiplied with the position of v as long as $\pi(u) < \pi(v)$. Therefore, we introduce a cost contribution function $c(v, \pi)$ for a vertex v in an arrangement π. Let $S \subseteq E$ be the set of edges that are incident to v, then

$$c(G, v, \pi) = \sum_{(u,v) \in S, \pi(u) < \pi(v)} w(u, v)$$

Note that $c(G, v, \pi)$ is not dependent on the exact position of v in π, but rather its relative position to its adjacent vertices in π. Therefore in $c(G, v, \pi)$ we do not multiply $w(u, v)$ with the position of v. The reason for this will become more apparent in Section 3.2.3.
Again, we reduce the notation to only $c(v, \pi)$ since π will often have a subscript of which graph/component it is an arrangement of. Figure 3.3 illustrates an arrangement π of a component and the weight of each edge is depicted directly above the edge. One could then see that $c(v_1, \pi) = 0$, $c(v_2, \pi) = 3$ and $c(v_3, \pi) = 5$.

Furthermore, for a component $G_i = (V', E')$ we calculate its cost contribution to $C(\pi)$ as the sum of the cost contribution of all vertices in G_i:

$$W(G_i, \pi) = \sum_{v \in V'} c(v, \pi)$$

Note that $W(G_i, \pi)$ is also not dependent on the exact position of each vertex but rather its relative position to the other vertices in G_i. The reason for this will become more apparent in Section 3.3.2.

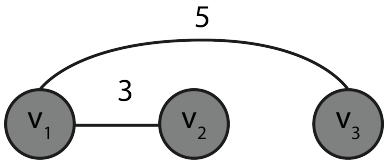


Figure 3.3: A linear arrangement π of a component.

3.1.5 Swaps

For an arrangement π a swap between two vertices u and v results in a new arrangement π' where $\pi'(u) = \pi(v)$ and $\pi'(v) = \pi(u)$. Furthermore, let S_1 and S_2 be two chains of G_i and G_j, where $i \neq j$, in π and assume that S_2 is positioned immediately to the right of S_1. Then, a swap between the two chains can be viewed as: removing S_1 from π and filling the empty spots by moving S_2 to the leftmost empty spot. After this, we place S_1 in the empty spots in π that S_2 has left behind. An example of this is illustrated in Figure 3.4, each step is numbered with a number at the top left corner. In Figure 3.4, the grey circles are positions that are occupied with a

vertex and the white circles are empty positions. Also, each vertex in each chains is labeled its original position in the chain. Step 1 illustrates the original state of π, step 2 illustrates the removal of S_2 and, finally, step 3 illustrates the new positions of the vertices S_2 and S_1.

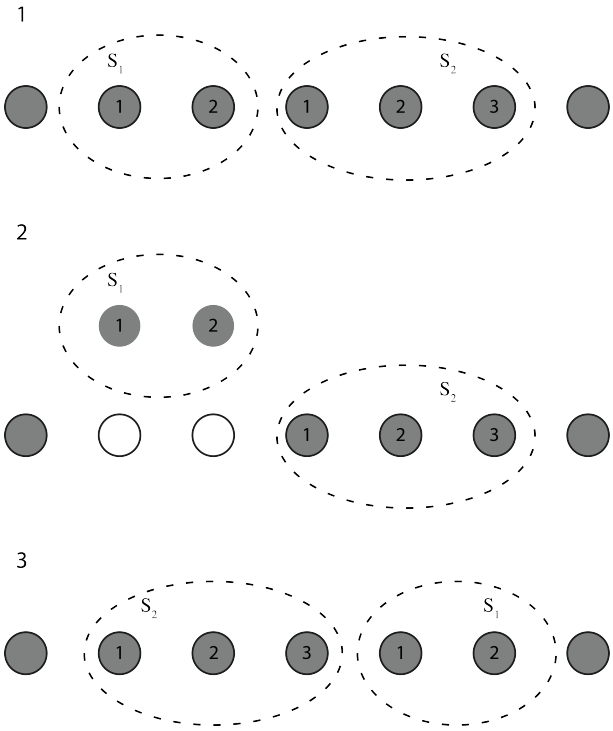


Figure 3.4: An illustration of a swap between two chains in a linear arrangement.

3.2 Optimal Arrangement of H

In this section we will present an algorithm that finds π^*_H in polynomial time given $\pi^*_{G_i}$ for each component in H.

We will start by introducing three lemmas for $\pi^*_{G_j}$ and in Sections 3.2.2−3.2.4 describe some procedures following those lemmas. Then, in Section 3.2.5 we describe some more lemmas that are vital for Algorithm 1 and finally in Section 3.2.6 we introduce the algorithm.

3.2.1 Properties of $\pi^*_{G_i}$ in π^*_H

In the following lemmas let $H = (V, E)$ denote a graph consisting of k components.

Lemma 3.2.1. *Let H' be an induced subgraph of H, consisting of l connected components, where l is chosen arbitrarily. Then, the structure of H' in π^*_H is the same as in $\pi^*_{H'}$.*

Proof. Suppose the structure of H' in π^*_H is not the same as in $\pi^*_{H'}$. Swapping the vertices of H' in π^*_H such that they have the same structure as in $\pi^*_{H'}$, decreases

$C(\pi^*_H)$. This is due to the fact that a swap between the vertices of H' in π^*_H does not affect vertices of $H \setminus H'$ and that $\pi^*_{H'}$ is an optimal arrangement of H'. □

Lemma 3.2.2. *The structure of G_i in π^*_H is the same as in $\pi^*_{G_i}$.*

Proof. Follows immediately from Lemma 3.2.1 with $l = 1$. □

Lemma 3.2.3. *Let v_1 and v_2 be two vertices of $G_i = (V', E')$, where $\pi^*_{G_i}(v_1) = 1$ and $\pi^*_{G_i}(v_2) = 2$. Then, v_1 and v_2 will be positioned immediately after each other in π^*_H.*

Proof. Following Lemma 3.2.2, if $\pi^*_{G_i}(v_1) = 1$ then v_1 is also the leftmost vertex of G_i in π^*_H. Now, assume $\pi^*_H(v_2) - \pi^*_H(v_1) > 1$, i.e. the distance between v_1 and v_2 in π^*_H is greater than one. We denote the set of vertices positioned between $\pi^*_H(v_1)$ and $\pi^*_H(v_2)$ as S. Remove v_1 from $\pi^*_H(v_1)$ and shift all vertices in S one step left, which leaves $\pi^*_H(v_2) - 1$ empty. Then, placing v_1 at $\pi^*_H(v_2) - 1$ we have a new arrangement π'_H where v_1 and v_2 are positioned successively. Since for all vertices $b \in S$, $\pi'_H(b) = \pi^*_H(b) - 1$, each b contributes less to $C(\pi'_H)$ than $C(\pi^*_H)$. Moreover, because v_1 is the leftmost vertex of G_i in both arrangements π^*_H and π'_H, $c(v_1, \pi^*_H) = 0 = c(v_1, \pi'_H)$, thus we have $C(\pi'_H) < C(\pi^*_H)$. Note, following Lemma 3.2.2, $S \cap V = \emptyset$ and therefore v_1 is still the leftmost vertex of G_i in π'_H. □

3.2.2 Unit Vertex

Let v_1 and v_2 be the two leftmost vertices of a component G_i in $\pi^*_{G_i}$. Lemma 3.2.3 suggests that v_1 and v_2 can be replaced with a single vertex U, which we will call a unit vertex. This is because v_1 and v_2 will also be positioned immediately after each other in π^*_H. Note that this might not be true for other vertices, for instance we cannot be certain that the three leftmost vertices of G_i in $\pi^*_{G_i}$ will also be positioned immediately after each other in π^*_H. Therefore only v_1 and v_2 are replaced with a unit vertex. We denote the graph where the two leftmost vertices of G_i, in $\pi^*_{G_i}$, are replaced with U as $G^\diamond_i$. Furthermore, we denote the graph H where each component $G_i \in H$ has replaced the two vertices that are positioned leftmost in $\pi^*_{G_i}$ with a unit vertex as $H^\diamond$. A replacement of v_1 and v_2 in G_i with U, where $\pi^*_{G_i}(v_1) = \pi^*_{G_i}(v_2) - 1$, is performed as follows:

1. Remove v_1 and v_2 from G_i and add a vertex U.
2. If there exists an edge (v_1, v_2) in G_i then, remove edge (v_1, v_2) and add a loop (U, U), where $w(U, U) = w(v_1, v_2)$.
3. For any vertex v adjacent to v_1 or v_2, remove the corresponding edge and add (U, v) where $w(U, v)$ is equal to two times the weight of the corresponding removed edge.

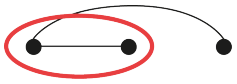
The reason for setting $w(U, v)$ to two times the weight of the corresponding edge in step 3 is explained in Section 3.2.3. Note that we are able to replace v_1 and v_2 with U because of Lemma 3.2.3, and since a unit vertex has a loop neither v_1 or v_2

themselves can be a unit vertex. Because, in such case Lemma 3.2.3 would not hold. Since we have $\pi^*_{G_i}$ we can obtain $\pi^*_{G_i^\diamond}$, as follows:

1. Set $\pi^*_{G_i}(U) = 1$ and remove v_1 and v_2 from $\pi^*_{G_i}$.
2. Move all vertices right of U in $\pi^*_{G_i}$ one step left.

At times we will need to check what type of vertex that occupies a position. Therefore, each vertex v in $H^\diamond$ gets a label v_* or U_* depending on if it is a regular vertex or unit vertex. We define a function $g(G, \pi_G, x)$ that returns the label of the vertex at position x in π. After finding $\pi^*_{H^\diamond}$ we can obtain π^*_H by filling an initially empty π_H as follows:

- For $j = 1, ..., N$, where N is the number of vertices in $H^\diamond$:
  - if $g(H^\diamond, \pi^*_{H^\diamond}, j) = v_*$ then place v at the leftmost empty position of π_H.
  - if $g(H^\diamond, \pi_{H^\diamond}, j) = U_*$ then place the corresponding v_1, v_2 at the leftmost empty position of π_H.



(a) A linear arrangement π_{G_i}. The two leftmost vertices, v_1 and v_2, in π_{G_i} are highlighted by a red circle.

(b) A linear arrangement $\pi_{G_i^\diamond}$. The leftmost vertex in $\pi_{G_i^\diamond}$ is a unit vertex.

Figure 3.5: The linear arrangement of G_i and $G_i^\diamond$

3.2.3 Cost of Swaps

In Section 3.1.5 we defined the execution of a swap, in this section we will describe how we can check if a swap is fruitful, i.e. decreases $C(\pi)$. We mainly consider swaps in π_H and $\pi_{H^\diamond}$, and following Lemma 3.2.2 there is no point in swapping vertices of the same component. Therefore we do not consider those.

For two consecutively positioned vertices u and v, where $\pi_H(u) < \pi_H(v)$, $C(\pi_H)$ decreases with $c(v, \pi_H)$ and increases with $c(u, \pi_H)$. Thus, if $c(v, \pi_H) > c(u, \pi_H)$ then a swap between the two is fruitful. Now we look at a swap between a regular vertex v and a unit vertex U, where v is positioned immediately left of U. Following the definition of unit vertices in Section 3.2.2, each U is simply a placeholder for two vertices. Therefore, in a swap between U and v, v actually moves 2 steps right and therefore $C(\pi_{H^\diamond})$ would increase with $2 \cdot c(v, \pi_{H^\diamond})$. However, we have compensated for this in step 3 in the replacement procedure in Section 3.2.2. Therefore, for a swap between U and v, $C(\pi_{H^\diamond})$ decreases if $c(U, \pi_{H^\diamond}) < c(v, \pi_{H^\diamond})$. Note that for a swap between two consecutively positioned vertices u and v in $\pi_{H^\diamond}$, we still only need to check if $c(v, \pi_{H^\diamond}) > c(u, \pi_{H^\diamond})$, since both vertices have twice their original cost contribution.

Let S_1 and S_2 be two chains of two different components in $\pi_{H^\diamond}$, where S_2 is positioned immediately right of S_1. Let $l_{1,v}$ be the number of vertices in S_2 and $l_{1,U}$ be the number of unit vertices in S_2. Then a swap between the two would increase $C(\pi_{H^\diamond})$ with:

$$(l_{1,v} + l_{1,U}) \cdot \sum_{v \in S_1} c(v, \pi_{H^\diamond}) + l_{1,v} \cdot \sum_{U \in S_1} c(U, \pi_{H^\diamond}) + l_{1,U} \cdot \sum_{U \in S_1} 2 \cdot c(U, \pi_{H^\diamond})$$

The same goes for the decrease in $C(\pi_{H^\diamond})$. Therefore, we introduce a function $P(H^\diamond, S_1, S_2, \pi_{H^\diamond})$ that calculates the total increase/decrease in $C(\pi_{H^\diamond})$, for a swap between S_1 and S_2, and returns the total amount of increase/decrease in $C(\pi_{H^\diamond})$. Once more, we reduce the notation to $P(S_1, S_2, \pi_{H^\diamond})$ since each π will have a subscript of the graph/component it is an arrangement for. Note that if $C(\pi_{H^\diamond})$ decreases in a swap between S_1 and S_2 then $P(S_1, S_2, \pi_{H^\diamond})$ will return a negative number.

Let us look at an example of how function $P()$ works. In Figure 3.6 we have depicted the arrangement $\pi_{H^\diamond}$, where $H^\diamond$ consists of two components $G_A^\diamond$ and $G_B^\diamond$. The vertices of $G_A^\diamond$ and $G_B^\diamond$ are depicted as red and blue circles respectively. Moreover, the cost contribution of each vertex is shown above respective vertex, for instance $c(v_{A,3}, \pi_{H^\diamond}) = 8$. Let the red dotted circle be the chain S_1 and let the blue dotted circle be the chain S_2. Also, assume $v_{B,1}$ is a unit vertex. We denote the number of regular vertices in S_1 as $l_{1,v}$ and we denote the number of regular and unit vertices in S_2 as $l_{2,v}$ and $l_{2,U}$ respectively. Then

$$\begin{aligned} P(S_1, S_2, \pi_{H^\diamond}) &= (l_{2,v} + l_{2,U}) \cdot c(v_{A,3}, \pi_{H^\diamond}) - l_{1,v} \cdot (c(v_{B,1}, \pi_{H^\diamond}) + c(v_{B,2}, \pi_{H^\diamond})) \\ &= (1+1) \cdot 8 - 1 \cdot (7 + 10) = -1 \end{aligned}$$

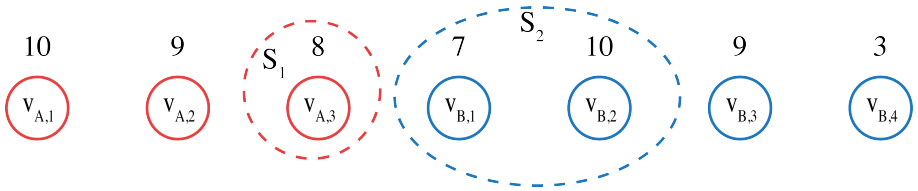


Figure 3.6: An example of $\pi_{H^\diamond}$. Edges are not depicted since they do not serve any purpose in the illustration.

3.2.4 Possible Arrangements

Let H be a graph consisting of two components G_A and G_B and let n_A and n_B be the number of vertices in G_A and G_B respectively. In π_H each vertex in G_B is positioned right/left of G_A or between two vertices of G_A. We will call these positions the relative positions to G_A since they are not fixed positions but rather a position relative to the vertices in G_A, see Figure 3.7. Following Lemma 3.2.3, in π_H^* no vertex in G_B is positioned between the two leftmost vertices in G_A and

vice versa. Furthermore, following Lemma 3.2.2, if we have an optimal arrangement of a component, for instance $\pi^*_{G_A}$ for G_A, then all possible relative positions to G_A are known. We denote each relative position as $a_{i,j}$ where i is the component that the relative position refers to and $j = 0, 2, 3, ..., n_i$ is the relative position from left to right of component G_i. We index each relative position such that the relative position immediately left of a vertex $v_{A,i}$ of G_A, where i indicates the position of the vertex in $\pi^*_{G_A}$, is $a_{A,i-1}$. Since the relative position between the two leftmost vertices of G_A is not an option there exists no $a_{A,1}$, instead there exist a relative position left of $v_{A,1}$, which is $a_{A,0}$. Consequently, there are n_A possible relative positions for each vertex in G_B, and various vertices in G_B can have the same relative position. Therefore, one can use the formula for combinations with repetition

$$\frac{(r+n-1)!}{r!(n-1)!}$$

and set $r = n_B - 1$ and $n = n_A$, to obtain the number of possible arrangements between G_A and G_B. Note that setting $r = n_A - 1$ and $n = n_B$ does not change the number of possible arrangements. This means that if we find an optimal relative position of a vertex v in G_B to G_A then we also know an optimal relative position of all vertices in G_A to v.

In Figure 3.7 an arrangement $\pi_{H^\diamond}$ is depicted. The red and blue circles represent the vertices of G_A and G_B, respectively. The curly brackets indicate the relative position between two vertices.

Figure 3.7: An arrangement $\pi_{H^\diamond}$. The edges of both components are left out because they do not serve any purpose in the illustration.

3.2.5 Optimal Relative Position

For an arbitrary vertex v a relative position $a_{i,j}$ is its unique optimal relative position (UORP) if $a_{i,j}$ is the relative position of v in all optimal arrangements of H. Moreover, if $a_{i,j}$ is an optimal relative position for v then there could not exist any other relative position for v such that $C(\pi_H)$ decreases if v is positioned at it. In this section we describe some properties of an optimal relative position for a vertex in $\pi^\diamond_H$ and these will later be used in Algorithm 1.

For the following lemmas let $L_A = \{v_{A,1}, ..., v_{A,l_A}\}$ and $L_B = \{v_{B,1}, ..., v_{B,l_B}\}$ be two chains of $G^\diamond_A$ and $G^\diamond_B$ in $\pi_{H^\diamond}$, where the second index of each vertex indicates its position in the chain. Note that since we are considering $\pi_{H^\diamond}$ the first vertex of a chain/subchain might be a unit vertex, however we will still denote it as v. Furthermore, let L_A be positioned immediately left of L_B and, following Lemma 3.2.2, the relative position of the vertices in each chain remains the same to the other vertices in the same chain.

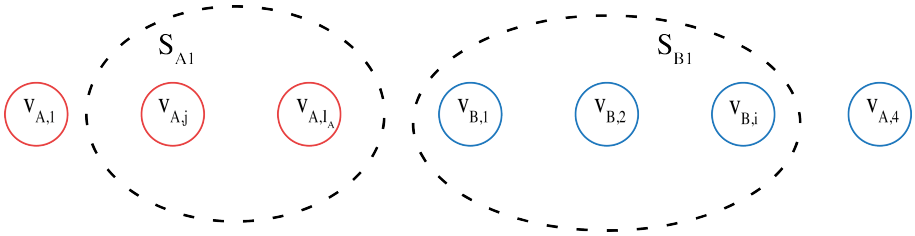


Figure 3.8: A possible scenario in Lemma 3.2.4 and 3.2.5. In this example the possible definitions for subchain S', from Lemma 3.2.4, are $S' = \{v_{B,1}\}$ and $S' = \{v_{B,1}, v_{B,2}\}$.

Lemma 3.2.4. *Let S_{A1} be a subchain of L_A from $v_{A,j}$ to v_{A,l_A}, where $j \leq l_A$. Let S_{B1} be a subchain of L_B, from $v_{B,1}$ to $v_{B,i}$, where $i \leq l_B$, see Figure 3.8. Assume there exists no subchain S' of $S_{B1} \setminus v_{B,i}$ from $v_{B,1}$, where $P(S', S_{A1}, \pi_{H^\diamond}) < 0$. If $P(S_{B1}, S_{A1}, \pi_{H^\diamond}) < 0$ then the current relative position of S_{B1}, that is a_{A,l_A}, is not an optimal relative position for any vertex in S_{B1}.*

Proof. Since the vertices in S_{B1} are positioned at a_{A,l_A}, if we swap such that all vertices in S_{B1} are positioned at $a_{A,j-1}$ then $C(\pi_{H^\diamond})$ decreases. Moreover, following the initial assumption, there exists no arrangement with a lower $C(\pi_{H^\diamond})$ where only a subchain S' of $S_{B,1} \setminus v_{B,i}$ is positioned at $a_{A,j-1}$ and the rest is positioned at a_{A,l_A}. Thus, a_{A,l_A} cannot be an optimal relative position for any vertex $S_{B,1}$. □

The assumption in Lemma 3.2.4 is of utmost importance for the lemma, else, following the notations in the lemma, one cannot be certain that for all possible subchains S' of $S_{B,1} \setminus v_{B,i}$, $P(S', S_{A1}, \pi_{H^\diamond}) > P(S_{B1}, S_{A1}, \pi_{H^\diamond})$. We will use this property in the following lemmas too, therefore we introduce the notion of inadequate vertices and inadequate chains/subchains.

Definition 3.2.1 (Inadequate)**.** Let S_{A1} be a subchain of L_A from $v_{A,j}$ to v_{A,l_A}. For a vertex $v_{B,i} \in L_B$, where $i \leq l_B$, we will call $v_{B,i}$ inadequate to the relative position $a_{A,j-1}$ if, for the subchain S_{B1} of L_B from $v_{B,1}$ to $v_{B,i}$, $P(S_{B1}, S_{A1}, \pi_{H^\diamond}) \geq 0$. Furthermore, we will call a chain/subchain S of a component, inadequate to a relative position $a_{i,j}$ if all vertices left of the rightmost vertex in S are inadequate to $a_{i,j}$.

With the help of this notion we will now extend Lemma 3.2.4 to a more general case.

Lemma 3.2.5. *Let S_{A1} and S_{B1} be the same chains as in Lemma 3.2.4. Assume that all vertices in L_B are inadequate to $a_{A,m}$ for $m = j, ..., l_A - 1$. Furthermore, assume that S_{B1} is inadequate to $a_{A,j-1}$. If $P(S_{B1}, S_{A1}, \pi_{H^\diamond}) < 0$ then no relative position to the right of $a_{A,j-1}$ is an optimal relative position for any vertex in S_{B1}.*

Proof. Following Lemma 3.2.4, a_{A,l_A} cannot be an optimal relative position. Moreover, let S_{B2} be a subchain of L_B from $v_{B,1}$. Since all vertices in L_B are inadequate to all relative positions between $a_{A,j-1}$ and a_{A,l_A}, $P(S_{B2}, S_{A2}, \pi_{H^\diamond}) \geq 0$ holds for all subchains S_{A2} of L_A from $v_{A,i}$ to v_{A,l_A}, where $j < i \leq l_A$. Thus, if the vertices in S_{B1}

are positioned to the right of $a_{A,j-1}$, swapping them such that they are positioned at $a_{A,j-1}$, decreases $C(\pi_{H^\diamond})$. Again, since S_{B1} is inadequate to $a_{A,j-1}$ there does not exist an arrangement with lower $C(\pi_{H^\diamond})$ where only a proper subset of S_{B1} is positioned at $a_{A,j-1}$ and the rest is positioned to the right of $a_{A,j-1}$. □

Lemma 3.2.6. *If all vertices in L_B are inadequate to $a_{A,i}$ then $a_{A,i}$ is not an UORP for any vertex in L_B.*

Proof. If $a_{A,i}$ is an UORP for any $v \in L_B$ then there exists no relative position for v to left or right of $a_{A,i}$ that either decreases $C(\pi_{H^\diamond})$ or $C(\pi_{H^\diamond})$ is the same. We prove that this does not hold for any $v \in L_B$. Let S_{A2} be subchain of L_A, from $v_{A,i+1}$ to v_{A,l_A}. We split the proof into two cases, one where $v = v_{B,1}$ and another one where $v = v_{B,j}$ for $j > 1$.
Case 1: $v = v_{B,1}$. Following the initial assumption, for all subchains S_{B1} of L_B from $v_{B,1}$, $P(S_{B1}, S_{A2}, \pi_{H^\diamond}) \geq 0$ holds. Thus, $C(\pi_{H^\diamond})$ will, at least, be the same regardless of whether v is positioned at its current relative position or at $a_{A,i}$.
Case 2: $v = v_{B,j}$. Let S_{B2} be a subchain of L_B, where $v \in S_{B2}$ and $v_{B,1} \notin S_{B2}$, and S_{B2} is inadequate to $a_{A,i}$. If there exists no subchain S_{B2} such that $P(S_{B2}, S_{A1}, \pi_{H^\diamond}) < 0$ then $a_{A,i}$ is not an UORP for v. Assume, instead, that there exists a subchain S_{B2} such that $P(S_{B2}, S_{A2}, \pi_{H^\diamond}) < 0$. Note that this does not violate our initial if statement. Let S_{B1} be the set of vertices in L_B positioned left of S_{B2}. Following the initial if statement, $P(S_{B1}, S_{A2}, \pi_{H^\diamond}) > 0$ else $P(S_{B1} \cup S_{B2}, S_{A2}, \pi_{H^\diamond}) < 0$, which violates the if statement. Moreover, following Lemma 3.2.2, all vertices in S_{B1} must have an optimal relative position at $a_{A,i}$ or left of it, for $a_{A,i}$ to be an UORP for the vertices in S_{B2}. We have already proven that $a_{A,i}$ cannot be an optimal relative position for vertices in $S_{B,1}$ since $P(S_{B1}, S_{A2}, \pi_{H^\diamond}) > 0$. Furthermore, for $S_{B,1}$ to have an optimal relative position left of $a_{A,i}$ there has to exist a subchain S_{A1} of L_A, from $v_{A,m}$ to v_{A,l_A}, where $m \leq i$, such that $P(S_{B1}, S_{A1}, \pi_{H^\diamond}) = 0$ or $P(S_{B1}, S_{A1}, \pi_{H^\diamond}) < 0$, Figure 3.9 illustrates an instance of this. If either one of the two scenarios occurs then

$$|S_{A2}| \cdot \sum_{v \in S_{A1} \setminus S_{A2}} c(v, \pi_{H^\diamond}) < |S_{A1} \setminus S_{A2}| \cdot \sum_{v \in S_{A2}} c(v, \pi_{H^\diamond})$$

where $|S_{A2}|$ is the number of vertices in S_{A2} and $|S_{A1} \setminus S_{A2}|$ is the number of vertices in S_{A1} that is positioned left of S_{A2}. Therefore, if all vertices in S_{B1} have an optimal relative position left of $a_{A,i}$ then there exists a relative position left of $a_{A,i}$ for all vertices in S_{B2} with a lower $C(\pi_{H^\diamond})$ than if S_{B2} is positioned at $a_{A,i}$. □

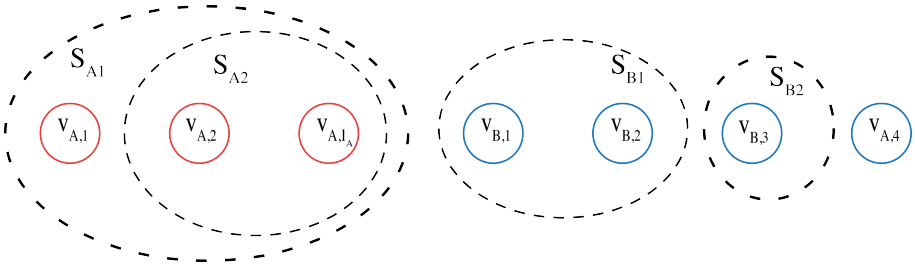


Figure 3.9: A possible scenario of Case 2 in Lemma 3.2.6

Lemma 3.2.7. *Let G_C be a third component in H and $L_C = \{v_{C,1}, ..., v_{C,l_C}\}$ a chain of $G_C^\diamond$ in $\pi_{H^\diamond}$. Assume that all vertices in $G_B^\diamond$ are in their optimal relative position to $G_A^\diamond$ and that a_{A,l_A} is an optimal relative position for all vertices in L_B. If all vertices in L_C are inadequate to $a_{B,m}$ for $m = i, ..., l_B$, where $a_{B,i}$ is the relative position left of L_B, then no relative position that is left of L_B is an UORP for any $v \in L_C$.*

Proof. Following Lemma 3.2.1, since $G_A^\diamond$ and $G_B^\diamond$ are optimally arranged to each other, L_B is positioned right of L_A in $\pi_{H^\diamond}$ too. For any relative position $a_{A,r}$, let S be the chain from $v_{A,r+1}$ to v_{A,l_A}. Since a_{A,l_A} is an optimal relative position for L_B, $P(S, L_B, \pi_{H^\diamond}) \geq 0$. Thus, for any subchain S_C of L_C from $v_{C,1}$, $P(S_C, L_B \cup S, \pi_{H^\diamond}) \geq 0$ holds. Furthermore, assume there exists vertices in $G_B^\diamond$ that are positioned left of L_B, we denote the closest chain of $G_B^\diamond$ left of L_B as L_{B2}. Note that there must exist a chain of $G_A^\diamond$ between L_B and L_{B2}, else L_{B2} would be part of L_B, we denote this chain as L_{A2}. Since L_{B2} is positioned at its optimal relative position to $G_A^\diamond$, $P(L_{B2}, L_{A2}, \pi_{H^\diamond}) \geq 0$ holds. Moreover, $P(L_B, L_{A2}, \pi_{H^\diamond}) \geq 0$ holds and thus $P(S_C, L_{B2} \cup L_{A2} \cup L_B, \pi_{H^\diamond}) \geq 0$. Therefore, no relative position left of L_B is an UORP for any vertex in L_C. □

3.2.6 Algorithm

Algorithm 1 finds an optimal arrangement of H given $\pi^*_{G_i}$ for each component in H. Following Lemma 3.2.2, each $\pi^*_{G_i}$ is "placed" in an initially empty π_H. This is done by positioning each $\pi^*_{G_i}$ at the leftmost empty positions in π_H where the structure of G_i in $\pi^*_{G_i}$ and π_H is the same. Below we will give a high-level description of the idea behind Algorithm 1 before defining the actual algorithm.

Algorithm 1 works similarly to insertion sort in that it goes from left to right in $\pi_{H^\diamond}$ and keeps an arranged left side $\pi_{LH^\diamond}$. At each step $\pi_{LH^\diamond}$ is extended to the next component and the optimal position for this component, in $\pi_{LH^\diamond}$, is found.

Algorithm 1.

1. Replace the two leftmost vertices of each G_i in $\pi^*_{G_i}$ with U_i and find $\pi^*_{G_i^\diamond}$ as described in Section 3.2.2.
2. Place each $\pi^*_{G_i^\diamond}$ in $\pi_{H^\diamond}$, in descending $W(G_i^\diamond, \pi^*_{G_i^\diamond})$ and assign a fixed index to each component from left to right. Let each unit vertex U have the same index as its corresponding component.
3. For $j = 2, ..., k$:
   - 3.1. Set pointer R_1 to $\pi_{H^\diamond}(U_j) - 1$, set $G_i^\diamond$ to the component that the vertex at R_1 belongs to. $G_i^\diamond$ is fixed until step 3.1 is revisited. Do:
     - 3.1.1. Set pointer R_2 to $\pi_{H^\diamond}(U_j)$.
     - 3.1.2. Set S_1 to the vertex at R_1 plus all vertices between R_1 and R_2. Moreover, set S_2 to the vertex at R_2 .
     - 3.1.3. If $P(S_2, S_1, \pi_{H^\diamond}) < 0$ then swap S_2 and S_1, and set $S_2 = \emptyset$.
     - 3.1.4. Move R_2 one step right, denote vertex at R_2 as u. If u belongs to $G_i^\diamond$ and a swap was conducted in step 3.1.3 then add u to S_1 and repeat step 3.1.4. If u belongs to $G_j^\diamond$ then add u to S_2 and go to step 3.1.3. Else go to step 3.2.
   - 3.2. Move R_1 one step left. If vertex at R_1 belongs to $G_i^\diamond$ then go to step 3.1.1, else if $R_1 + 1 = \pi_{H^\diamond}(U_j)$ go to 3.1. Else, Go to next iteration.
4. Expand $\pi_{H^\diamond}$ to π_H.

Theorem 3.2.8. *Given $\pi^*_{G_i}$ for each component, Algorithm 1 finds an optimal arrangement π^*_H of H in polynomial time.*

Proof. Going through the algorithm, step 1 can be motivated by Lemma 3.2.2 and 3.2.3. Moreover, step 2 reduces the number of swaps since for any $\pi^*_{G_j^\diamond}$ placed initially right of another $\pi^*_{G_i^\diamond}$ there exists no unique optimal arrangement where all vertices in $\pi^*_{G_j^\diamond}$ are placed left of $\pi^*_{G_i^\diamond}$, an example of this is shown in Section 3.3.2. Now, we will motivate the loop at step 3.

Let j have the same definition as in the loop at step 3. Let q be the total number of vertices in the first j components. We denote the arrangement $\pi_{H^\diamond}$ from 0 to q as $\pi_{LH^\diamond}$. Note that j increases at each iteration. Following Lemma 3.2.1, if we find the optimal relative position of the j first component then we also know their relative position to each other in $\pi_{H^\diamond}$. Thus once an optimal relative position of $G_j^\diamond$ in $\pi_{LH^\diamond}$ is found, $G_j^\diamond$ does not need to change its relative position to the other components that where already in $\pi_{LH^\diamond}$. At each iteration the algorithm swaps each $v \in G_j^\diamond$ to left until one of the two possible cases occur:

Case 1 : For each $v \in G_j^\diamond$: The rightmost component $G_i^\diamond$ in $\pi_{LH^\diamond}$, where an optimal relative position for v to $G_i^\diamond$ is $a_{i,m}$ such that $m > 0$, is found.

Case 2 : When $\pi_{LH^\diamond}(U_j) = 1$ is the optimal position for U_j.

We start by proving that if one of the two cases occur then we have found an optimal relative position for each $v \in G_j^\diamond$ in $\pi_{LH^\diamond}$. After this, we prove that the algorithm correctly finds one of the two cases for each $v \in G_j^\diamond$.

Case 1 : Following Lemma 3.2.1, if $a_{i,m}$ is an optimal relative position for v to $G_i^\diamond$ and $G_i^\diamond$ is optimally arranged in $\pi_{LH^\diamond}$ then there exists no UORP left of $a_{i,m}$ for v and thus it is its optimal relative position in $\pi_{LH^\diamond}$.

Case 2 : Since $G_j^\diamond$ is initially the rightmost component in $\pi_{LH^\diamond}$ and we swap each $v \in G_j^\diamond$ to left, except for U_j in $G_j^\diamond$, either case 1 has occurred or $a_{i,0}$ is an optimal relative position for v to all other components in $\pi_{LH^\diamond}$. Thus, $G_j^\diamond$ is optimally arranged in $\pi_{LH^\diamond}$.

Now we prove that the algorithm correctly finds one of the two cases for each $v \in G_j^\diamond$.

Let L_j be a chain of $G_j^\diamond$ and let $L_i = \{v_r, ..., v_{l_i}\}$ be a chain of an arbitrary component $G_i^\diamond$ in $\pi_{LH^\diamond}$, where $r \leq n_i$ and n_i the number of vertices in $G_i^\diamond$. Moreover, let L_i be positioned immediately left of L_j. Note that there might exist vertices in $G_i^\diamond$ positioned left of L_i with vertices of other components in-between L_i and the other vertices in $G_i^\diamond$. However, following Lemma 3.2.7, if an optimal relative position for $v \in L_j$ to L_i is $a_{i,m}$, where $m > r - 1$, then, $a_{i,m}$ is also an optimal relative position for v to $G_i^\diamond$.

There is another loop from steps $3.1 - 3.2$, we will call this the inner loop. Moreover, let S_1 be the same S_1 as in the inner loop. At each iteration of the inner loop it is checked if each $v \in L_j$ is at an optimal relative position or not. Following Lemma 3.2.5, if a swap is possible then v is not in an optimal relative position and therefore it is swapped to a position such that $C(\pi_{H^\diamond})$ decreases. Furthermore, following Lemma 3.2.6, the relative position immediately left of S_1, in an inner loop iteration, is not an UORP for any of the vertices in L_j that was not swapped. Let S_{j1} be the subchain of L_j that is swapped in an inner loop iteration and let S_{j2} be the subchain of L_j that is not swapped. For each $v \in S_{j2}$, since it was not swapped, $C(\pi_{H^\diamond})$ is at least the same whether v is positioned at the relative position immediately left of S_1 or at its current position. Therefore, if S_{j2} is not swapped to any other relative position to L_i then, S_{j2} is in an optimal relative position to L_i. Moreover, since S_{j1} was swapped and S_{j2} was not, this means that

$$|S_{j1}| \cdot \sum_{v \in S_{j2}} c(v, \pi_{H^\diamond}) < |S_{j2}| \cdot \sum_{v \in S_{j1}} c(v, \pi_{H^\diamond})$$

Then, for each relative position to L_i if the rightmost vertex in S_{j1} is inadequate to it then so are all vertices in S_{j2}. Thus following this procedure one if the two cases will eventually occur. □

Complexity analysis of Algorithm 1. We assume each $\pi^*_{G_i}$ is found in polynomial time, both step 1 and 2 take $O(nk)$ time, where n is the number of vertices in the component with most vertices. Moreover, at each iteration $G_j^\diamond$ is compared, at

most, to all components initially to its left, thus the inner loop is run $O(k^2)$ times. Finding an optimal arrangement between two components $G_i^\diamond, G_j^\diamond$ we have to check all n_i possible relative positions for all $n_j - 1$ vertices in G_j thus there are $O(n^2)$ checks for finding the optimal relative position of $G_j^\diamond$ to $G_i^\diamond$. At steps 3.1.4 and 3.2 it is checked if a vertex u belongs to a certain component $G_i^\diamond$. We now give a simple suggestion on how this can be done in step 3.2 in $O(\log nk)$ time, the same approach can also be applied to step 3.1.4. Let v_i be the leftmost vertex in S_1 and let v the vertex such that $\pi^*_{G_i^\diamond}(v) = \pi^*_{G_i^\diamond}(v_i) - 1$. If u belongs to G_i then u should be the vertex position immediately left of v_i in $\pi^*_{G_i^\diamond}$. Therefore, if $\pi_{LH^\diamond}(u) = \pi_{LH^\diamond}(v)$ then $u = v$ and thus u belongs to $G_i^\diamond$. In total the algorithm runs in $O(k^2 n^2 \log nk)$.

Corollary 3.2.1. *We allow vertices in H to have loops. Given $\pi^*_{G_i}$ for each component H, the optimal arrangement of H can be found in polynomial time.*

Proof. Lemma 3.2.3, does no longer hold for all components, because the leftmost vertex v of a $\pi^*_{G_i}$ now might have a loop. Thus only if the leftmost vertex in a $\pi^*_{G_i}$ does not have a loop do we replace it with a unit vertex. Note that the second leftmost vertex in any $\pi^*_{G_i}$ might have a loop but the weight of this edge can simply be added to the edge (U_i, U_i). Apart from this, loops do not affect the outcome of Algorithm 1 and thus they can be solved in polynomial time. □

3.3 Trees

In this section we describe a property for the arrangement of trees and we describe how one can reduce the number of unique optimal arrangements between two trees.

In Section 3.3.1 we introduce a lemma for the arrangement of trees. With the help of this lemma, we prove that Algorithm 2, that is basically the G2 algorithm in [3], finds an optimal arrangement for a tree of, at most, order three in linear time. Then, using Algorithm 2 and Algorithm 1 we solve the problem on forests consisting of trees of, at most, order three in polynomial time. Finally, in Section 3.3.2 we show an example of how the number of unique optimal arrangements between two trees T_A and T_B are reduced when placing them in decreasing $W(T_i, \pi^*_{T_i})$.

3.3.1 Property for the Arrangement of a Tree

Lemma 3.3.1. *For a tree $T = (V, E)$ there exists an optimal arrangement π^*_T where each $v \in V$, where $d(v) = 1$, is positioned to the right of its adjacent vertex u, where $d(u) > 1$.*

Proof. Let $u \in V$ be a vertex such that u has at least one adjacent vertex $v \in V$ where $d(v) = 1$. Moreover, we denote the set of adjacent vertices to u as $N \subseteq V$. Assume we have an optimal arrangement π^*_T where u is positioned to the right of a $v \in N$, where $d(v) = 1$. We can swap u and v in π^*_T and obtain a new arrangement π'_T. The edge (u, v) in π'_T has the same position as in π^*_T and for all other vertices $b \in N$ the edges (u, b) either have a lower end than before or the same thus $C(\pi'_T) \leq C(\pi^*_T)$. □

We now present the $G2$ algorithm introduced by Boysen and Stephan [3], with a slight modification, for a tree $T = (V, E)$ where $|V| \leq 3$. We name it Algorithm 2 so that it follows the naming convention of this thesis. Note that this is a very trivial case and one could simply compare all possible arrangement instead to find the optimal arrangement of T. However, the purpose of Algorithm 2 is to demonstrate Lemma 3.3.1 and to, together with Algorithm 1, present a polynomial solution for forests consisting of trees of at most order three.

Algorithm 2. Given a tree $T = (V, E)$ where $|V| \leq 3$, for each vertex $v_i \in V$ set $x_i = \sum_{(v_i,u)\in E} w(v_i, u)$. Arrange each $v_i \in V$ in π_T such that x_i decreases going from left to right in π_T.

Theorem 3.3.2. *Algorithm 2 returns an optimal linear arrangement for a tree $T = (V, E)$ where $|V| \leq 3$.*

Proof. Follows immediately from Lemma 3.3.1. □

This algorithm is basically the $G2$ algorithm presented by Boysen and Stephan, but instead of arranging based on the number of incident edges of each vertex we calculate x_i for each vertex. Now, using this algorithm one can find the optimal arrangement of each tree in a forest consisting of trees of, at most, order three. Then using Algorithm 1 one can obtain the optimal arrangement of the forest in polynomial time.

3.3.2 Candidate Arrangements

Let F be a forest consisting of two trees. Following Lemma 3.2.3 and 3.2.2 there exists six possible arrangements of the two trees in F, these arrangements are depicted in Figure 3.10. We denote the red and black tree, in Figure 3.10, as T_A and T_B respectively. The unit vertex of each tree is represented as a square and each arrangement in Figure 3.10 is numerated. We will refer to arrangement 1 as $\pi^1_{F^\diamond}$ and so on.

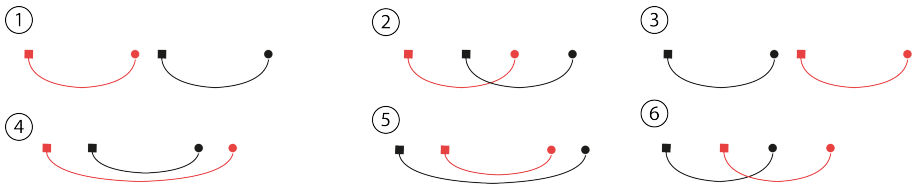


Figure 3.10: Possible arrangements of two trees. Squares represent unit vertices and circles regular vertices. The loop of each unit vertex is not depicted.

Theorem 3.3.3. *If $W(T_A, \pi^*_{T_A}) \geq W(T_B, \pi^*_{T_B})$, then $\pi^3_{F^\diamond}$ and $\pi^6_{F^\diamond}$ cannot be the unique optimal arrangement of F.*

Proof. We split the proof into two cases, one for $\pi^3_{F^\diamond}$ and one for $\pi^6_{F^\diamond}$.
Case 1: Assume $\pi^3_{F^\diamond}$ is a unique solution to F. Then $W(T_A, \pi^*_{T_A}) < W(T_B, \pi^*_{T_B})$, which violates our initial if statement.

Case 2 : Assume $\pi^6_{F^\diamond}$ is an unique solution to F. Then $c(U_B, \pi_F) > c(U_A, \pi_F)$, else we could swap U_B and U_A, and $c(v_B, \pi_F) > c(v_A, \pi_F)$, else v_A and v_B could be swapped. Thus, $W(T_A, \pi^*_{T_A}) < W(T_B, \pi^*_{T_B})$ which violates our first if statement. □

From Lemma 3.3.3 we can draw the conclusion that for two trees arranged in decreasing $W(T_i, \pi^*_{T_i})$ order it suffices to check $\pi^1_{F^\diamond}$, $\pi^2_{F^\diamond}$, $\pi^4_{F^\diamond}$ and $\pi^5_{F^\diamond}$.

4

Discussion

In this chapter we discuss the results and suggest directions for future work.

4.1 Results

Algorithm 1 proves that, given $\pi^*_{G_i}$ for each component in a graph H, the optimal arrangement of H can be found in polynomial time. We note that for many components finding their optimal arrangement might be NP-hard. For graphs that consists of such components Algorithm 1 will not be effective since finding the optimal arrangement of a such component will have a greater time complexity than polynomial, unless $P = NP$. However, for the components that are solvable in polynomial time the algorithm enables graphs consisting of such components to be solved in polynomial time. This reduces the problem on graphs with more than one component to only finding the optimal arrangement of each component individually. Furthermore, this result can increase the understanding of the complexity of the product location problem on a single rack. Having said that, Algorithm 1 could probably be improved to have a better time complexity and the proof for the algorithm could probably be more straightforward by only considering the effects of $c(v, \pi_H)$ for each vertex in the arrangement of H.

The results of this thesis are not directly applicable to real-world warehouses since the orders' in real-world warehouses often consist of more than two products. Also, the layout in real-world warehouses are often not as simple as in the case we treated in this thesis. Hypergraphs are better representation of the orders' since each edge can consist of various number vertices. We strongly believe that a similar approach as in Algorithm 1 could find the optimal arrangement of a weighted hypergraph where the optimal arrangement of each set of disjoint edges are given.

Algorithm 1 has not been implemented in any programming language, but we believe that it can be implemented in code without too much struggle. We believe that the most abstract part of Algorithm 1 is determining if a vertex at a specific position belongs to a component, this is done at steps 3.1.4 and 3.2 in the Algorithm 1. However, we have proposed a procedure for this in the Complexity analysis in Section 3.2.6, that can be easily translated to actual code.

4.2 Future Work

A promising direction for future work would be to investigate if a similar approach as in Algorithm 1 would solve the product placement problem for weighted hypergraphs where the optimal arrangement of each disjoint set of edges is given.

As mentioned in previous section, finding the optimal arrangement of many components might be NP-hard. Therefore one might wonder how Algorithm 1 performs given a suboptimal arrangement for each component in H. We have not investigated how Algorithm 1 would perform in such situations and this would be an interesting question to investigate.

Another interesting direction would be to consider some Branch and Bound solution for small components, i.e. components with few edges. Branch and bound is an algorithm technique mostly used for solving optimization problems, a loose description of the technique is that it performs a breadth first search through the solution space and chooses the solution with the best result. Usually, this technique leads to an exponential solution, but if one can find a good lower or upper bound on the problem then one can discard a solution that exceeds these bounds and thus obtain better solutions. In our case this would mean that if we can discard some possible arrangements then the solution space reduces and the Branch and Bound technique could obtain a better solution. In Section 3.3.2 and in Lemma 3.3.1 we describe two such properties that reduce the number of possible solutions.

5

Conclusion

In this thesis we have examined the product location problem on a pick-by-order policy restricted to a single rack with front depot and restricted to, at most, two products in each order. We introduced a polynomial algorithm for this problem on an arbitrary weighted graph given the optimal arrangement of each component. Moreover, we showed how Algorithm 1 together with Algorithm 2 can solve the problem for forests consisting of trees of, at most, order three in polynomial time.

Bibliography

[1] Assmann, S. F., Peck, G. W., Sysło, M. M., and Zak, J. (1981). *The bandwidth of caterpillars with hairs of length 1 and 2*. SIAM Journal on Algebraic Discrete Methods, 2(4), 387-393.

[2] Boysen, N., Füßler, D., and Stephan, K. (2020). *See the light: Optimization of put-to-light order picking systems*. Naval Research Logistics (NRL), 67(1), 3-20.

[3] Boysen, Nils, and Konrad Stephan. (2013) *The deterministic product location problem under a pick-by-order policy.* Discrete Applied Mathematics 161.18 : 2862-2875.

[4] Díaz, J., Petit, J., and Serna, M. (2002). *A survey of graph layout problems*. ACM Computing Surveys (CSUR), 34(3), 313-356.

[5] Held, M., and Karp, R. M. (1962). *A dynamic programming approach to sequencing problems*. Journal of the Society for Industrial and Applied mathematics, 10(1), 196-210.

[6] Lin, M., Lin, Z., and Xu, J. (2006). *Graph bandwidth of weighted caterpillars*. Theoretical computer science, 363(3), 266-277.

[7] Monien, B. (1986). *The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete*. SIAM Journal on Algebraic Discrete Methods, 7(4), 505-512.

[8] Papadimitriou, C. H. (1976). *The NP-completeness of the bandwidth minimization problem*. Computing, 16(3), 263-270.

[9] Van Oudheusden, D. L., and Zhu, W. (1992) *Storage layout of AS/RS racks based on recurrent orders*. European journal of operational research, 58(1), 48-56.