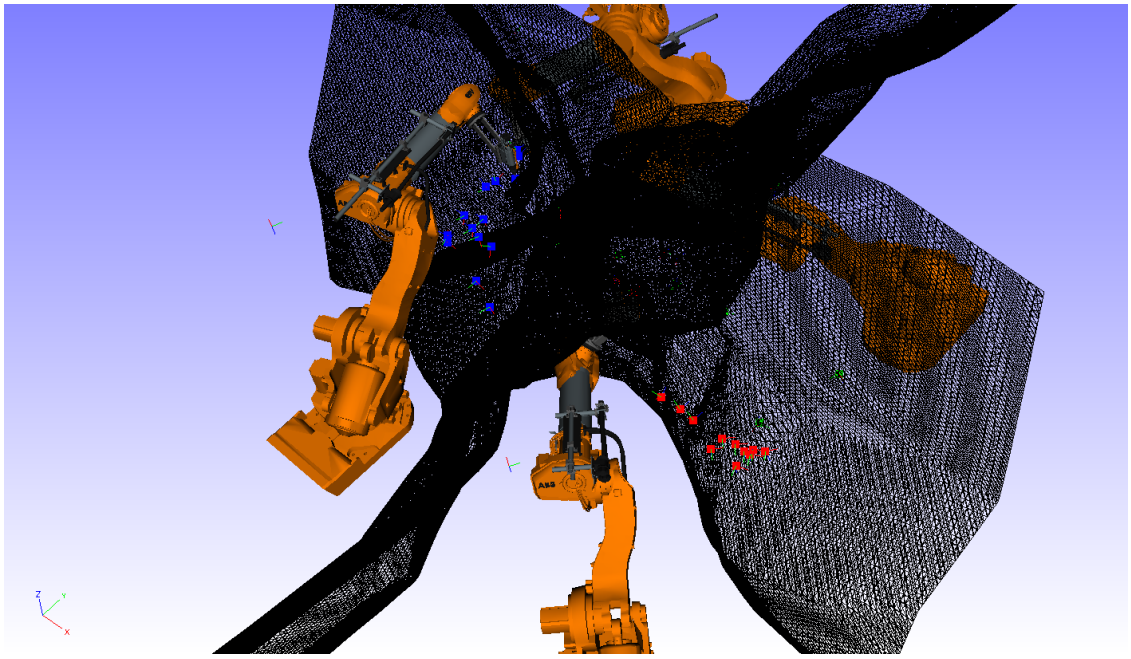


Intersection-free load balancing for industrial robots

Modelling and algorithm development

Master's thesis in Engineering Mathematics and Computational Science

EDVIN ÅBLAD

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Master's thesis 2016

Intersection-free load balancing for industrial robots

Modelling and algorithm development

Edvin Åblad

UNIVERSITY OF GOTHENBURG

CHALMERS

Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2016

Intersection-free load balancing for industrial robots
Modelling and algorithm development
Edvin Åblad

Supervisor: Domenico Spensieri, Fraunhofer-Chalmers Centre
Examiner: Ann-Brith Strömberg, Chalmers University of Technology

Master's Thesis 2016
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg

Cover: A robot station partitioned into four separate cells; one for each robot.

Typeset in LaTeX
Gothenburg, Sweden 2016

Intersection-free load balancing for industrial robots
Modelling and algorithm development
Edvin Åblad
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis considers the problem of minimizing cycle time while avoiding collisions within a robot station consisting of multiple industrial robots that collectively should cover a set of tasks, e.g. welding operations, on a workpiece. The cycle time is the time required to perform all tasks. Nowadays, in order to prevent the robots from colliding, their programs usually synchronise by adding synchronisation/interlocking signals when necessary. Here, the aim is instead to partition the space within the station, separating the robots. There are three main advantages in this approach: -the simplicity of the robot programs due to the total absence of synchronisation need; -the stability of the station in case of unexpected events; -lower maintenance cost. The problem is thus to find a space partition allowing all tasks to be performed in a minimal time. In order to retrieve the space partition, an approximate problem is repeatedly solved using the Dantzig-Wolfe decomposition principle and from the solutions provided, generalised Voronoi diagrams are approximated. For each of the resulting candidate partitions the robot module in the software IPS is used to determine the robot station cycle time. Results on existing industrial test cases show that using this approach the cycle time was increased by around 5% as compared with synchronising the robot programs. It is however not determined whether this approach finds an optimal partitioning since the generalised Voronoi diagram is generated from some stationary position of the robots, which might cause non-optimal paths between these positions.

Keywords: column generation, Dantzig-Wolfe decomposition, generalised Voronoi diagram, satisfiability problem, space partitioning of robot programs.

Acknowledgements

Several individuals have contributed greatly in the creation of this thesis; the main co-contributor is my supervisor Domenico Spensieri that gave many useful advises regarding almost every part of the algorithm. My examiner Ann-Brith Strömberg, from Chalmers University of Technology, Johan S. Carlson and Robert Bohlin, from Fraunhofer-Chalmers Centre (FCC), also contributed by giving regular directions of the project. Jonas Kressin, Johan Nyström & Tomas Hermansson, from FCC, gave support in how to use the IPS software. Ann-Brith & Domenico also provided much help in the writing of this thesis.

Edvin Åblad, Gothenburg, June 2016

Contents

1

Introduction

1.1 Background

Fraunhofer-Chalmers Centre is a research centre, located at the Johanneberg Science Park, focusing on modelling, simulation and optimization of products and processes with the aim to boost technical development, improve efficiency and cut costs for several industrial areas: among these, an important sector is the automotive industry, where its manufacturing process is of special interest (see [1]).

The manufacturing process of vehicles is a complex process composed by many steps and advanced components. One of the final steps in the process is the steel metal sheet assembly. Here, often a workpiece, as a car body, is handled by several robots operating in the same workstation. Robots can perform welding tasks on specific points of the workpiece, they can seal, or measure predefined objects' areas, using laser or other techniques.

Optimizing the throughput of such stations is crucial not only to meet the demands of the automotive market but also to ensure both the economical and the ecological sustainability of the production; in fact, the system becomes more efficient in terms of energy and space, see [2].

1.2 Motivation

Industrial robots' motions are programmed to avoid collisions with the environment. Synchronisation messages, exchanged between the robots and a *programmable logic controller* (PLC), are added to the programs, in order to avoid collisions also among the robots (see [3]). This additional step decreases the station's flexibility, since the synchronisations are integrated into the robot programs: changing a program would often require checking whether the synchronisation scheme is still valid. Maintenance costs are also increased, due to the longer times needed for replacing a malfunctioning robot and for moving the robots home in a safe way after a sudden production stop (see [4]). Motivated by the above observations, this thesis work will consider the possibility to decrease the complexity of the robot programs while keeping cycle time under an acceptable threshold value. The objective is to generate collision-free robot programs that perform all the specified tasks on the workpiece, minimizes cycle time, and under the constraint that the volumes swept along the robots' paths do not intersect.

1.3 Previous work

Much of the previous work on the problem addressed in this work relies on the fact that if the issue of colliding robots is initially disregarded, the problem can be modelled as a min-max generalised multiple travelling salesperson problem (min-max GMTSP) or as an uncapacitated min–max Generalised Vehicle Routing Problem. In a min-max GMTSP, multiple agents collectively cover all the clusters in a graph while minimising the length of the longest tour; it generalises the multiple travelling salesperson problem (MTSP) since the agents may have different starting nodes and since the nodes are grouped into clusters and it is enough to visit one node in each cluster. The collision handing is typically done in a post-processing step, in which the robot programmes are synchronised; see [5, 3, 6]. For an overview of the travelling salesperson problem; see [7, Ch. 1].

1.4 Limitations

Since the algorithm developed to generate the collisions-free robot programs contains many components, many of which are part of existing software, the mathematical theory behind these components are not presented in this work but only referred to. However, for the two main parts of the algorithm to which this thesis work contributes the most: (i) the solution of the min-max semi-assignment problem with a conflict constraint, and (ii) the generalised Voronoi diagram, some mathematical background and alternative solution methods are presented and tested.

1.5 Problem introduction

Figure 1.1 illustrates a robot station. In this particular station, four robots collectively need to perform 48 stud weld tasks on the car body; a stud weld is technique

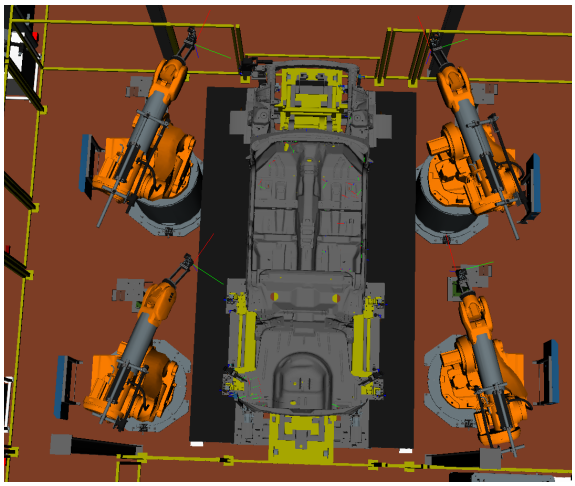
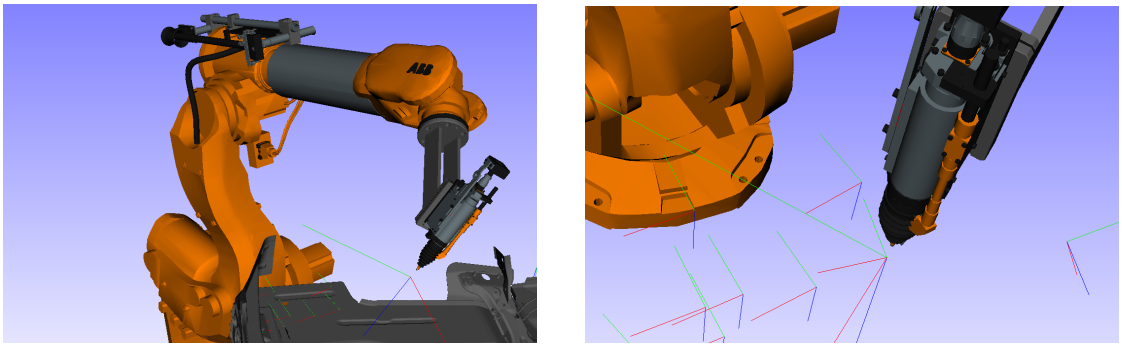


Figure 1.1: A snapshot from IPS illustrating four robots in a robot station, a car body (that is the workpiece), and 48 stud weld tasks on the workpiece. A close up of a robot is shown in Figure 1.2a and a close up of a task is shown in Figure 1.2b.

to fasten a nut. The typical goal is to minimise the station's cycle time or makespan, i.e., the time of the longest robot cycle, which is equivalent to maximising the station's throughput, i.e. the rate of production [8, p. 570]. Figure 1.1 is a snapshot from the software Industrial Path Solutions (IPS), which offers sequencing and path planning in a robot station; see Section 4.1.2 for more details.

In Figure 1.2a, the geometry of a robot is visualised. This robot is a typical industrial robot with six joints: rotation around the base plate, backward & forward tilting at the base plate, lowering and raising arm, rotating arm, lowering and raising the tool, and rotating the tool. This type of robot is considered throughout the thesis but the algorithm developed may be extended to handle other types of robots.



(a) A robot positioned near the car body.

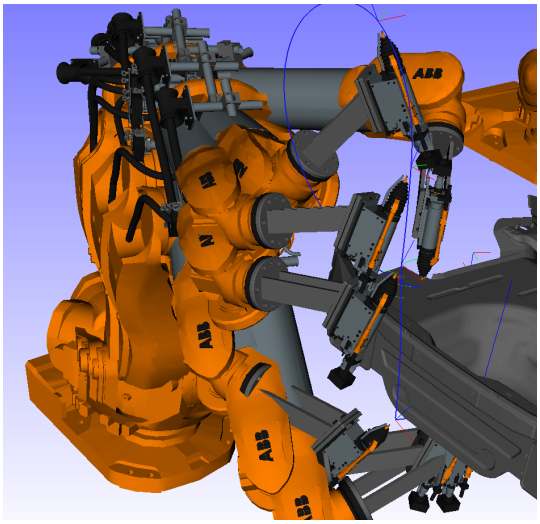
(b) A robot positioned at a task.

Figure 1.2: Illustrations of details in the robot station shown in Figure 1.1. In order to emphasise the robot and the tasks, the surrounding geometry as well as the car body (Figure 1.2b) are not rendered.

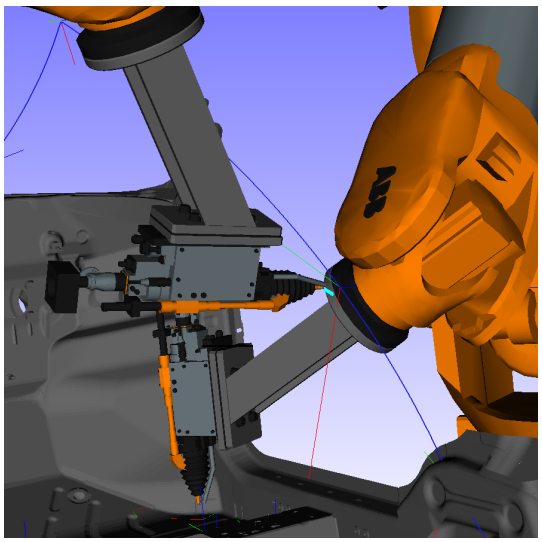
Figure 1.2b illustrates a close-up on a cluster of stud weld tasks. Note that a stud weld can be performed with any rotation in the plane where the stud is to be placed; this is reflected in the Figure where the tool centre point (TCP) aligns in the plane (blue line) but with a rotation (red and green lines). Note that there are other types of tasks: restrictive tasks where the TCP needs a specific rotation and general tasks where the TCP follows a path when performing the task. Therefore, throughout this thesis, we assume that there are several alternative robot configurations that can be used to perform each task. Hence, any type of task that is performed using a stationary robot position is considered.

As mentioned in Section 1.2, the problem to solve in such a station is to assign every task to a robot, order the tasks in a sequence, and find a path through this sequence (in the right order). In Figure 1.3a illustrates a shortest path between two tasks; the robot must keep a certain clearance to the car body to guarantee a collision free path. This clearance is often a sum of many contributions: the uncertainties in the geometry of the car body, the inaccuracy of the robot's motion, and of other process specific requirements.

In Figure 1.3b, the issue of a robot-robot collision is visualised, i.e., there is no guarantee that the robots will not collide during this robot program since a robot-robot clearance limit is not met. Planning all the robots' moves simultaneously to find collision-free motions would resolve this issue. Anyway, in IPS this approach is



(a) A collision free path, where the robot is sampled several times to illustrate the motion along the path.



(b) A robot-robot collision, where the shortest distance between the robots is 4cm and marked by a cyan thick line.

Figure 1.3: Illustrate the robot's motion along paths between tasks, where the path is marked by a blue curve denoting the position of the robot TCP along the path. The paths guarantee a certain clearance to any static geometry but not to other robot objects.

not adopted since the complexity of such algorithms grows exponentially with the number of robots and that would be prohibitive. Several decoupled strategies have been devised to cope with this problem (see [6]) and currently IPS handles this issue by synchronising the robots using coordination signals in the robot programs.

In order to remove the need for synchronisation between robots the additional constraint—that the volumes swept along the robots' paths pass the desired clearance check— is added. Note that is equivalent with Definition 1.5.1. This equivalence can be established by two facts: (i) given such a partition, the constraint is trivially satisfied, (ii) the existence of one such partition is provided by the generalised Voronoi diagram, introduced in Section 2.7. This constraint is later called the *robot-partitioning constraint* or simply the *partitioning constraint*.

Definition 1.5.1 (robot-partitioning constraint). This constraint is satisfied if the space can be partitioned into subsets, one for each robot, such that the swept volume of each robot is strictly contained in the corresponding subset with half the robot-robot clearance to the subset boundary.

Having defined the robot-partitioning constraint we are now able to define the *complete problem*, the goal of this thesis is thus to construct an algorithm that provides good solutions to this problem. However, before presenting the model and the algorithm in Section 3 some mathematical tools will be introduced, in Section 2.

Definition 1.5.2 (complete problem). Generate robot programs with minimal

the cycle time, performs all tasks, has a clearance to the environment, and satisfies the robot-partitioning constraint (recall Definition 1.5.1).

2

Mathematical and computational tools

We next present some key concepts. The algorithm in Section 3 considers an integer linear program with constraints in conjunctive normal form; see Section 2.1. Solving this integer linear program involves column generation—see Section 2.2— used in a Dantzig-Wolfe decomposition (see Section 2.3) and a branch-and-price procedure to find integral solutions (see Section 2.4). The algorithm also involves a partitioning of the robots using an approximation of the generalised Voronoi diagram (Sections 2.5 & 2.7). In an attempt to improve the approximation, a Delaunay tessellation is used; see Section 2.6. A final component of the algorithm requires a solution to a GTSP, in addition the cost of the edges in this problem is costly to compute, to solve this problem the software IPS is used; see Section 2.8.

2.1 Conjunctive normal form

For a collection of Boolean variables (or constants) with members a_k where $k = 1, \ldots, n$ for some n. Let C_{ij} be an atomic formula, i.e. either a_k or $\neg a_k$ for some k determined by i and j. An expression is said to be in *conjunctive normal form* (CNF) if it is on the form

$$\bigwedge_i \bigvee_j C_{ij} \tag{2.1}$$

and $\wedge$ is the *and* operator, known as a conjunction, and $\vee$ is the *or* operator, known as a disjunction. This form may seem restrictive but in fact every propositional formula can be converted to a CNF expression; see [9]. Here, we will discuss two logical expressions that will be of later use.

***xor*(a, b)** denotes the *exclusive or* function and returns *true* when either a or b but not both are *true*. Hence, we have the equivalent expressions

$$(a \vee b) \wedge \neg(a \wedge b) \Leftrightarrow (a \vee b) \wedge (\neg a \vee \neg b), \tag{2.2}$$

where the right expression is in CNF and the equivalence is established by De Morgan's laws (see [10]).

n*-*xor*$(a_1, a_2, \ldots, a_n)$** is an extension of the ***xor function, which returns *true* when exactly one a_i is *true*. A CNF expression for this is

$$\left(\bigvee_{i=1}^{n} a_i\right) \wedge \left(\bigwedge_{1 \geq i < n} \bigwedge_{i < j \geq n} (\neg a_i \vee \neg a_j)\right), \tag{2.3}$$

where the first disjunction expresses that at least one a_i is *true* and the remaining disjunctions expresses that at most one a_i is *true*. Note that the number of terms in (2.3) equals $1 + \frac{1}{2}n(n-1)$.

2.2 Column generation

Here, we follow the procedure in [11, Ch. 3.3]. Suppose we have the general linear programming (LP) problem

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimise}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x}, && \text{(2.4a)} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, && \text{(2.4b)} \\ & \mathbf{x} \geq \mathbf{0}. && \text{(2.4c)} \end{aligned}$$

where the number n of variables (i.e., $\mathbf{x} \in \mathbb{R}^n$) is huge. Considering all of them might be hard. Instead, we consider only a subset of p variables, denoted by tilde, as $\tilde{\mathbf{x}} \in \mathbb{R}^p$. Given an optimal solution $\tilde{\mathbf{x}}^*$ to the corresponding restricted problem

$$\begin{aligned} \underset{\tilde{\mathbf{x}}}{\text{minimise}} \quad & \tilde{\mathbf{c}}^{\mathrm{T}}\tilde{\mathbf{x}}, && \text{(2.5a)} \\ \text{s.t.} \quad & \tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, && \text{(2.5b)} \\ & \tilde{\mathbf{x}} \geq \mathbf{0}, && \text{(2.5c)} \end{aligned}$$

where tilde denote the corresponding subset of $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{c}$, two questions are relevant:

- Is the solution $\tilde{\mathbf{x}}^*$ optimal in the original problem (2.4)?
- If not, which variables can be included in the subset to improve the solution?

These questions are related to the simplex algorithm, which is a general algorithm for solving LP problems; see [12, Ch. 4]. The simplex algorithm is based on the fact that an optimal solution to problem (2.4) will be in an extreme point of (2.4b). One way to express these extreme points $\mathbf{x}$, is to first solve

$$\mathbf{x}_{\mathbf{B}} = \mathbf{B}^{-1}\mathbf{b}, \tag{2.6}$$

where $\mathbf{B}$ is a matrix of m linearly independent columns of $\mathbf{A}$ and m is the number of rows of $\mathbf{A}$. The extreme point $\mathbf{x}$ has the values of $\mathbf{x}_{\mathbf{B}}$ on indexes corresponding to the selected columns of $\mathbf{A}$ and is zero otherwise. Let $\mathbf{c}_{\mathbf{B}}$ be a vector with elements of $\mathbf{c}$ corresponding to the vector $x_{\mathbf{B}}$. With these we may express the row vector of simplex multipliers $\boldsymbol{\pi} = \mathbf{c}_{\mathrm{B}}^{\mathrm{T}}\mathbf{B}^{-1}$, which are also called LP dual variables. The simplex algorithm then uses the concept of reduced cost

$$\bar{\mathbf{c}} := \mathbf{c} - \mathbf{A}^{\mathrm{T}}\boldsymbol{\pi}, \tag{2.7}$$

which is the marginal gain of using others columns of $\mathbf{A}$. Hence, for an element i with negative $\bar{\mathbf{c}}_i$ the solution will improve by switching to an extreme point that

use the i-th column of $\mathbf{A}$. Note that for all already selected columns of $\mathbf{A}$, the corresponding element $\bar{\mathbf{c}}_i$ is zero.

Using the reduced cost, we may return to the two above questions. First assume that an optimal solution $\tilde{\mathbf{x}}^*$ to the restricted problem (2.5) is found and denote the corresponding multipliers by $\bar{\boldsymbol{\pi}}$. If all reduced costs (2.7) are positive, then the solution $\tilde{\mathbf{x}}^*$ is optimal to the complete problem. Otherwise, the idea behind column generation, is to introduce one or several such variables (columns) with negative reduced cost into the restricted problem and continue iteratively by checking for optimality.

Consider a slightly different problem, where the constraints (2.4b) are relaxed to $\mathbf{Ax} \leq \mathbf{b}$. By adding slack variables $\mathbf{s}$, we regain a problem on the form (2.4), where the constraint matrix $\mathbf{A}$ is replaced by $[\mathbf{A}\ \mathbf{I}]$. After the introduction of the slack variables, the expression (2.7) for the reduced costs still holds. Now by including all slack variables in the restricted problem (2.5) there is no need to compute the reduced cost for the slack variables.

The procedure, of finding the column corresponding to the lowest reduced cost, and introducing it in the subset, also provides bounds on the optimal value, which can be used in a termination criterion. The upper bound is trivial, i.e. the value of an optimal solution to the restricted problem will always be equal to or higher than that of the solution to the problem containing all columns. The lower bound, as mentioned by [13], can be computed when an upper bound, κ, to the column sum $\mathbf{1}^{\mathrm{T}}\mathbf{x}$ is given for any $\mathbf{x}$ feasible in (2.4). Then [13] states that the solution cannot improve more than κ times the smallest reduced cost. Summarising, the bounds on the optimal value z^* in terms of the optimal value to the restricted problem, i.e. $\bar{z}$, and the smallest reduced cost, $\bar{c}^*$, is given by

$$\bar{z} + \kappa\bar{c}^* \leq z^* \leq \bar{z}. \tag{2.8}$$

2.3 Dantzig-Wolfe decomposition

This Section follows the procedure in [14, pp. 319–320], which is a special case of the so-called Dantzig-Wolfe decomposition. A more general treatment is found in [11, Ch. 3]. Consider an optimisation problem of the following form, where $\mathbf{A}$ denotes an $m \times n$ matrix:

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimise}} \quad & c(\mathbf{x}), && \text{(2.9a)}\\
\text{s.t.} \quad & \mathbf{Ax} = \mathbf{b}, && \text{(2.9b)}\\
& \mathbf{x} \in \mathbf{S}, && \text{(2.9c)}\\
& \mathbf{x} \ \text{ integer}. && \text{(2.9d)}
\end{aligned}$$

Note that the function $c(\mathbf{x})$ is not necessarily linear nor convex, although it simplifies the problem greatly, since in this method the problem (2.9a), (2.9c), & (2.9d) needs to be solved. In this thesis, $\mathbf{S}$ is assumed to be a convex and bounded polyhedron; for the unbounded case, see e.g. [15]. By using this assumption, the set

$$\mathbf{S}^* = \{\mathbf{x} \in \mathbf{S} \colon \mathbf{x} \ \text{ integer}\} \tag{2.10}$$

is a finite set of vectors, $\{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$. Hence, any point in $\mathbf{S}^*$ may be expressed as

$$\mathbf{x} = \sum_{j=1}^{M} \lambda_j \mathbf{x}^j, \qquad \text{such that} \qquad \sum_{j=1}^{M} \lambda_j = 1 \qquad \text{and} \qquad \lambda_j \in \{0, 1\}. \tag{2.11}$$

By inserting (2.11) into (2.9) the following column generation form of (2.9):

$$\begin{aligned}
\underset{\lambda_j}{\text{minimise}} \quad & \sum_{j=1}^{M} c(\mathbf{x}^j)\lambda_j, && \text{(2.12a)}\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mathbf{A}\mathbf{x}^j \lambda_j = \mathbf{b}, && \text{(2.12b)}\\
& \sum_{j=1}^{M} \lambda_j = 1, && \text{(2.12c)}\\
& \lambda_j \in \{0, 1\}, \qquad j = 1, \ldots, M. && \text{(2.12d)}
\end{aligned}$$

Note that $c_j := c(\mathbf{x}^j)$ and $\mathbf{p}_j := (\mathbf{A}\mathbf{x}^j)$ are constants in this new equivalent formulation and that λ_j are variables.

Note that the possibly non-linear integer optimisation problem (2.9) has been transformed into the integer linear programming (ILP) problem (2.12), here referred to as the integer master problem (IMP). However, since the set $\mathbf{S}^*$ is in general very large, the number M of variables in (2.12) is also very large. By relaxing the integrality on $\boldsymbol{\lambda}$ (the constraint (2.12d)) to $\lambda_j \in [0, 1]$, we retrieve the so-called master problem (MP):

$$\begin{aligned}
\underset{\lambda_j}{\text{minimise}} \quad & \sum_{j=1}^{M} c(\mathbf{x}^j)\lambda_j, && \text{(2.13a)}\\
\text{s.t.} \quad & \text{(2.12b) and (2.12c) hold}, && \text{(2.13b)}\\
& \lambda_j \in [0, 1], \qquad j = 1, \ldots, M. && \text{(2.13c)}
\end{aligned}$$

As noted in Section (2.2) we can solve this problem using the column generation principle, i.e., only considering a subset of $\mathbf{S}^*$ in (2.12) (then represented by a subset of the set $\{1, \ldots, M\}$): this is called the restricted master problem (RMP). Given an optimal solution to the RMP with the LP dual optimal variable values $\bar{\boldsymbol{\pi}}$ and $\bar{q}$ corresponding to the constraints (2.12b) and (2.12c), respectively. This yield, according to (2.7), the reduced cost of a point in the set $\mathbf{S}^*$, which in this case becomes,

$$\bar{c}^j = c(\mathbf{x}^j) - \bar{\boldsymbol{\pi}}^{\mathrm{T}} \mathbf{A}\mathbf{x}^j - \bar{q}. \tag{2.14}$$

Hence, if we find a solution to the problem to

$$\underset{\mathbf{x} \in \mathbf{S}^*}{\text{minimise}} \quad c(\mathbf{x}) - \bar{\boldsymbol{\pi}}^{\mathrm{T}} \mathbf{A}\mathbf{x} - \bar{q}, \tag{2.15}$$

with a negative objective value, introducing the corresponding column in the RMP will improve its objective value. Otherwise —if the optimal objective value of (2.15) is non-negative— the solution to the RMP also solves the MP. Note that since $\bar{\boldsymbol{\pi}}$

and $\bar{q}$ are optimal LP dual variables to the RMP, every column in the RMP has non-negative reduced cost.

This procedure is a decomposition procedure, since, if $\mathbf{S}$ and c can be separated, then the subproblem will separate into smaller subproblems. To be more precise, if the set S can be expressed as $\mathbf{S} = \prod_{1 \leq i \leq k} \mathbf{S}_i$, where $\mathbf{S}_i^* = \{\mathbf{x} \in \mathbf{S}_i \colon \mathbf{x} \text{ integer}\}$ with elements $\mathbf{x}^{i1}, \ldots, \mathbf{x}^{iM_i}$, and c may be decomposed, i.e.

$$c(\mathbf{x}^j) = \sum_{i=1}^{k} c(\mathbf{x}^{ij}), \qquad \text{for any} \qquad \mathbf{x}^j = \sum_{i=1}^{k} \mathbf{x}^{ij},\ \mathbf{x}^{ij} \in \mathbf{S}_i^*, \tag{2.16}$$

which holds, e.g., if c is linear. Then the problem may be decomposed using the relations in (2.16) and representing the points in $\mathbf{S}_i$ using the convexity formulation (recall eq. (2.11)) the model (2.12) becomes

$$\underset{\lambda_{ij}}{\text{minimise}} \quad \sum_{i=1}^{k} \sum_{j=1}^{M_i} \Big(c(\mathbf{x}^{ij})\Big)\lambda_{ij}, \tag{2.17a}$$

$$\text{s.t.} \quad \sum_{i=1}^{k} \sum_{j=1}^{M_i} \Big(\mathbf{A}\mathbf{x}^{ij}\Big)\lambda_{ij} = \mathbf{b}, \tag{2.17b}$$

$$\sum_{j=1}^{M_i} \lambda_{ij} = 1, \qquad i = 1, \ldots, k, \tag{2.17c}$$

$$\lambda_{ij} \in \{0, 1\}, \quad j = 1, \ldots, M_i,\ i = 1, \ldots, k. \tag{2.17d}$$

The model (2.17) is also an ILP problem. Hence, if the sets $\mathbf{S}_i^*$ are too large then we may again use the column generation principle by relaxing the integrality on λ_{ij}. Using the relation (2.16) and noting that the coefficients of the reduced cost (2.7) for λ_{ij} are only dependent on members of $\mathbf{S}_i^*$, the subproblem of finding negative reduced cost decomposes over the sets $\mathbf{S}_i^*$, and can be expressed as to

$$\underset{\mathbf{x} \in \mathbf{S}_i^*}{\text{minimise}}\ c(\mathbf{x}) - \bar{\boldsymbol{\pi}}^{\mathrm{T}} \mathbf{A}\mathbf{x} - \bar{\mathbf{q}}_i. \tag{2.18}$$

A natural question that arises: if this procedure still requires a continuous relaxation of the convexity variables λ_j, then we still need to do some procedure to retrieve the integral solution; why not simply relax the integrality on $\mathbf{x}$ in the original problem (2.9)? The answer is that the lower bounds received from the MP is no worse (but possibly better) than the bounds obtained by a linear relaxation of the original problem. Moreover, it is the same bound as retrieved from a Lagrangian relaxation with respect to the constraints (2.9b); see [16].

Deriving a lower bound for the MP is a simple task —recall that the lower bound in the general column generation situation is given by (2.8) and that the column sum is bounded by the convexity constraint— the bounds become

$$z^* \geq \bar{z} + \bar{c}^* \tag{2.19}$$

for the undecomposed case. For the decomposed case this become

$$z^* \geq \bar{z} + \sum_{i=1}^{k} \bar{c}_i^*, \tag{2.20}$$

since the cost function c decompose according to (2.16). Note that these bounds hold true also for the inequality constraints in (2.9b).

The careful reader might have noticed that in order to compute $\bar{\boldsymbol{\pi}}$ and $\bar{q}$ in the RMP, it must contain at least one column and, for the optimal dual objective to be finite, the column must be a feasible solution to the MP. This initial column may be found using a phase 1 of the simplex algorithm (see [12, Ch. 4.9]) or some problem specific heuristic that is able to determine whether the problem is infeasible.

It remains to satisfy the integrality condition in the IMP (2.12). In order to satisfy this condition a branch-and-price procedure is employed, as presented below.

2.4 Branch-and-price

For the purpose of this work, it is sufficient to branch only on the original variables (i.e., $\mathbf{x}$). This type of branching and more advanced branchings are explained in [17]. The procedure works as follows: when the MP (2.13) is solved, the values of the original variables $\mathbf{x}$ are computed from the optimal values of the convexity variables λ_j^*, using (2.11). If the value of some element in $\mathbf{x}$ is fractional, then the branching rule is applied. What branching rule to use depends on the problem at hand but generally we partition $\mathbb{R}^n$ into B and B^c, then two branches are created by restricting the set $\mathbf{S}^*$ to be either $\mathbf{S}^* \cap B$ or $\mathbf{S}^* \cap B^c$; hence, the RMP remains unchanged but the subproblem(s) will be minimised over these new sets. As noted in [18] this is a preferable approach if the subproblem(s) remain tractable.

In each node of the branching, the solution of the MP does not need to be started from scratch, since earlier nodes might have generated good columns. Hence, the concept of a column pool is introduced, and in each node it is checked which columns in the pool are feasible under the current branching rule. Any columns found during this procedure is added to the pool. There is a risk for the column pool to grow too large and thus several approaches have been developed for deleting columns from the pool: the simplest of these might be that if the pool becomes too large all columns are deleted before the column generation begins, due to time constraint this simple rule was implemented; for better rules see [14].

A node is cut off from the tree if either the branching rule allows no feasible solution to the problem (2.9), or the lower bound from the MP under the branching rule exceeds the value of the best integer solution found so far.

Whenever the solution to the MP happens to be fractional in a node, and the node is not cut off from the tree, the branching rule is applied. An example of the branch-and-price procedure is illustrated in Figure 2.1. The details of the branching rule will be discussed in Section 3.3.7.

2.5 Voronoi diagram

Here the concept of Voronoi diagram is introduced with the purpose of defining the generalised Voronoi diagram, which in the case of a robot station can be informally described as the surface that separates all robots and is everywhere equidistant to

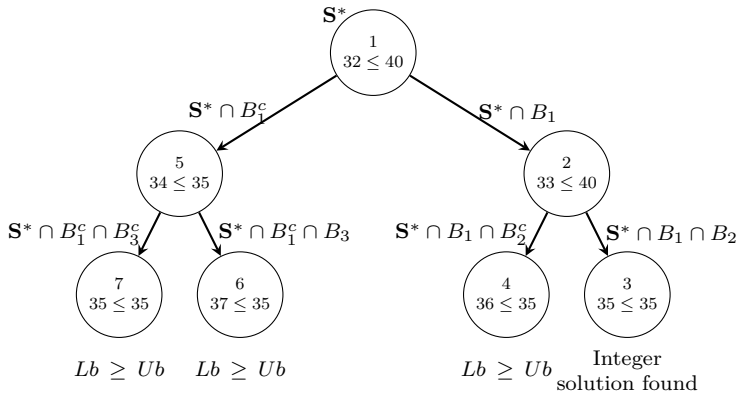


Figure 2.1: Example of a branch-and-price tree, where in each node the MP is solved for the set indicated above the node. The $Lb \leq Ub$ given in each node denotes the value of the solution found to the MP and that of the best integer solution found; the number above this denotes the order in which the nodes are visited.

the two closest robots. This Section contains mainly a selection of the work done in [19], which is also recommended for further details.

The concept of Voronoi diagram is ancient, due to its frequent occurrence in nature, and there are very early publications, e.g. [20] from 1644, where it is used to describe distribution of matter in the solar system.

The ordinary Voronoi diagram is defined for a set, P, of distinct points in $\mathbb{R}^m$; these are often called generator points since they generate a Voronoi cell. The Voronoi cell generated by a point in P is a subset of $\mathbb{R}^m$, which in the Euclidean distance measure is closer to the generator point than to any other point in the set P. A more precise definition is given next.

Definition 2.5.1 (Voronoi diagram in $\mathbb{R}^m$ [19, p. 45]). Let $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset \mathbb{R}^m$ be a finite set of distinct points. The set

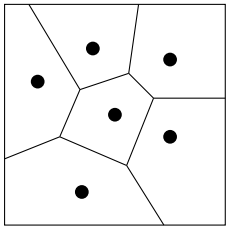
$$V(\mathbf{p}_i) := \{\mathbf{x} \in \mathbb{R}^m \colon \|\mathbf{x} - \mathbf{p}_i\| \leq \|\mathbf{x} - \mathbf{p}_j\|, \quad j \in \{1, \ldots, n\}\} \tag{2.21}$$

is called the m-dimensional Voronoi cell generated by $\mathbf{p}_i$, the plane shared by two Voronoi cells is called a Voronoi face, and the extreme points of a Voronoi cell are called Voronoi vertices. The set of all Voronoi cells $\mathcal{V}(P) = \{V(\mathbf{p}_1), \ldots, V(\mathbf{p}_n)\}$, is called the m-dimensional Voronoi diagram generated by P.

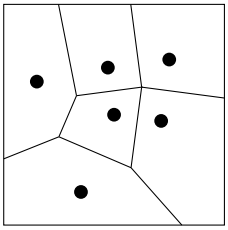
An alternative to (2.21) is to use the concept of half-spaces (denoted H), which emphasise the linearity of the faces in the Voronoi diagram, according to

$$V(\mathbf{p}_i) = \bigcap_{\in P \setminus \{i\}} H(\mathbf{p}_i, \mathbf{p}_j), \quad H(\mathbf{p}_i, \mathbf{p}_j) := \{\mathbf{x} \in \mathbb{R}^m \colon \|\mathbf{x} - \mathbf{p}_i\| \leq \|\mathbf{x} - \mathbf{p}_j\|\}. \tag{2.22}$$

The half-space $H(\mathbf{p}_i, \mathbf{p}_j)$, can be explicitly expressed in terms of scalar products rather than norms as $\mathbf{n}_{ij} \cdot \mathbf{x} \geq b_{ij}$, where $\mathbf{n}_{ij} := \mathbf{p}_i - \mathbf{p}_j$ denotes the normal of the half-space, and $b_{ij} := \frac{1}{2}(\mathbf{p}_i - \mathbf{p}_j) \cdot (\mathbf{p}_i + \mathbf{p}_j)$ denotes its level.



(a) A non-degenerate Voronoi diagram.



(b) A degenerate Voronoi diagram.

Figure 2.2: Two examples of planar ($m = 2$) Voronoi diagrams generated by six points. Note the interpretation of (2.22); each cell is the intersection of every half-space generated by the cell generator point together with any other generator point.

An example of a Voronoi diagram in the plane is presented in Figure 2.2a.

An important property of the Voronoi diagram is that it is unique by construction; see Property 2.5.1. This is good to keep in mind when the dual of the Voronoi diagram, the Delaunay tessellation, is introduced in Section 2.6.

Property 2.5.1 (Uniqueness of Voronoi diagram [19, p. 58]). Given $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset \mathbb{R}^m$ of distinct points. Let $V(\mathbf{p}_i)$ be given by (2.21), which is a non-empty, convex m-dimensional polyhedron. $\mathcal{V}(P)$ is a unique tessellation of $\mathbb{R}^m$, since it covers $\mathbb{R}^m$ (2.23a) and contains no overlaps (2.23b).

$$\bigcup_{i=1}^{n} V(\mathbf{p}_i) = \mathbb{R}^m, \tag{2.23a}$$

$$\{V(\mathbf{p}_i) \setminus \partial V(\mathbf{p}_i)\} \cap V(\mathbf{p}_j) = \emptyset, \qquad i \in \{1, \ldots, n\} \setminus \{j\}, j \in \{1, \ldots, n\}, \tag{2.23b}$$

where ∂ denote the set boundary operator. □

Another important and related property is whether the Voronoi diagram is non-degenerate, which is of importance for the uniqueness of the Delaunay tessellation.

Definition 2.5.2 (Non-degenerate Voronoi diagram [21, p. 2]). An m-dimensional Voronoi diagram is called non-degenerate if each Voronoi vertex is contained in exactly $m + 1$ Voronoi cells. □

An example of a degenerate Voronoi diagram is shown in Figure 2.2b, where the computed Voronoi cells reveal that one Voronoi vertex is contain in more than three cells.

2.6 Delaunay tessellation

In the algorithm suggested, in Section 3.4.3, which assigns a specific volume of the station to each robot, the Voronoi diagram is central. The Delaunay tessellation is

only used in a suggested improvement of the algorithm; thus this Section is optional for understanding.

The m-dimensional Delaunay tessellation[1] of a point set $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset \mathbb{R}^m$, is a set of simplices with vertices in P, covering the convex hull of P, and having no overlapping members. See Figure 2.3a for examples in two dimensions. We next use the Voronoi diagram to define the Delaunay tessellation, and will use this definition to prove the empty circum-hypersphere property (which is sometimes used to define the Delaunay tessellation [21, p. 1]).

Definition 2.6.1 (m-dimensional Delaunay tessellation [19, p. 56]). Let

- $P := \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset R^m$ be a set of unique points,
- $\mathcal{V}(P)$ denote its Voronoi diagram,
- $T := \text{conv}(P)$ possess non-zero volume,
- $Q := \{\mathbf{q}_1, \ldots, \mathbf{q}_l\}$ denote the set of Voronoi vertices in $\mathcal{V}(P)$,
- $P_i := \{\mathbf{p}_{i1}, \ldots, \mathbf{p}_{ik_i}\}$ denote the set of generator points of Voronoi cells containing Voronoi vertex $\mathbf{q}_i$,
- $T_i := \text{conv}(P_i)$, and
- $\mathcal{D}(P) := \{T_1, \ldots, T_l\}$.

If $k_i = m + 1$ for all $i \in \{1, \ldots, l\}$ then $\mathcal{D}(P)$ consists of l m-dimensional simplices and it is the m-dimensional Delaunay tessellation of T, which spans the set P.

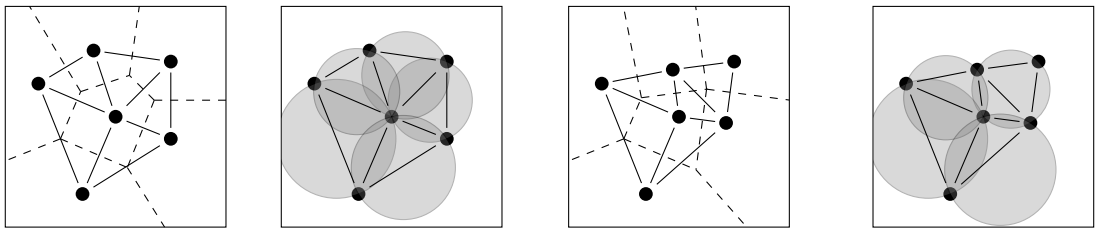
If there exists an $i \in \{1, \ldots, l\}$ such that $k_i \geq m + 2$, then we call the set $\mathcal{D}(P)$ the m-dimensional Delaunay pretessellation of T, which spans P. The Delaunay tessellation, consisting of m-dimensional simplices, is retrieved from the pretessellation by partitioning each T_i, for which it holds that $k_i \geq m + 2$, into $k_i - m$ simplices, by non-intersecting hyperplanes passing through the vertices of T_i. □

The m-dimensional simplices in the Delaunay tessellation will be referred to as Delaunay simplices. Note that if the Voronoi diagram $\mathcal{V}(P)$ is non-degenerate then, by Definition 2.5.2, we have that $k_i = m + 1$ for all $i \in \{1, \ldots, l\}$ and from the uniqueness of the Voronoi diagram the Delaunay tessellation must be unique; see Figure 2.3a. However, if the Voronoi diagram is degenerate then we note that there exists an $i \in \{1, \ldots, l\}$ such that $k_i \geq m + 2$ and hence the pretessellation will be partitioned until a Delaunay tessellation is achieved; since this partition can be done in multiple ways (compare Figure 2.3c) the resulting Delaunay tessellation is not unique. We conclude that a Delaunay tessellation is unique whenever the Voronoi diagram is non-degenerate.

[1] The Delaunay tessellation is known as the Delaunay triangulation in two dimensions or Delaunay tetrahedralization in three dimensions.

Property 2.6.1 (Empty circum-hypersphere of Delaunay simplices [19, p. 74]). Every Delaunay simplex will have an circum-hypersphere, which will be centred at the Voronoi vertex shared by all generator points in the simplex and the circum-hypersphere will be empty (strictly containing no generator point).

Since the Voronoi vertex is by definition equally close to all generator points in the simplex, this distance defines the radius of the circum-hypersphere. Now assume that a generating point is closer to the Voronoi vertex than this radius, which implies that the vertex cannot be contained in any Voronoi cell generated by the points in the simplex. This leads to a contradiction since the Voronoi vertex is on the boundary of all these Voronoi cells, hence the circum-hypersphere must be empty. □



(a) Delaunay triangulation

(b) Circumcircles of the triangulation

(c) Delaunay triangulation in the degenerate case

(d) Circumcircles in the degenerate case

Figure 2.3: Illustrations of Delaunay triangulations for two different point sets. (a) and (b) illustrate the triangulation (solid lines), of the first point set: (a) the Voronoi diagram (dashed); (b) the empty circumcircles. (c) and (d) illustrate a degenerate Voronoi diagram: in (c) a Voronoi vertex is shared by four Voronoi cells; the resulting Delaunay triangulation is randomly determined; in (d) it is shown how these four generator points are located on the same empty circumcircle.

To conclude this Section, we note that the Delaunay tessellation is widely used in applications (see [22]) and in the 2D-case it is considered the optimal triangulation in many aspects. In higher dimensions it still possesses many positive attributes, e.g. the empty circum-hypersphere property. These properties have made the Delaunay tessellation popular. For more details on the mathematical properties on the Delaunay tessellation see [23]. For 3D applications, it is however not practice to use the Delaunay tetrahedralization but rather some extension of it, this to ensure certain mesh qualities that is not guaranteed by the Delaunay tetrahedralization, as in [24].

2.7 Generalised Voronoi diagram

The Voronoi diagram (VD) can be extended in several ways; see [19]. A common and useful generalisation is to replace the point set P, employed in Definition 2.5.1 with a finite set of pairwise disjoint objects $\mathcal{A}$. Definition 2.7.1 is a simplification of

that suggested in [19, p. 115–118]; this is appropriate for our application to a robot station, for which the general definition is not necessary.

Definition 2.7.1 (Generalised Voronoi diagram (GVD)). Given a space $\mathbb{S}$, equipped with a distance metric $d\colon \mathbb{S} \times \mathbb{S} \to \mathbb{R}$. Given a finite set of pairwise disjoint generator objects $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, where $A_i \subseteq \mathbb{S}$, the Voronoi cell corresponding to A_i is given by

$$V(A_i) = \left\{ \mathbf{x} \in \mathbb{S} \colon \inf_{\mathbf{a} \in A_i} d(\mathbf{a}, \mathbf{x}) \leq \inf_{\mathbf{a} \in A_j} d(\mathbf{a}, \mathbf{x}),\ j = 1, \ldots, n \right\}$$

and the GVD generated by $\mathcal{A}$ is the set $\mathcal{V}(\mathcal{A}) = \{V(A_1), \ldots, V(A_n)\}$.

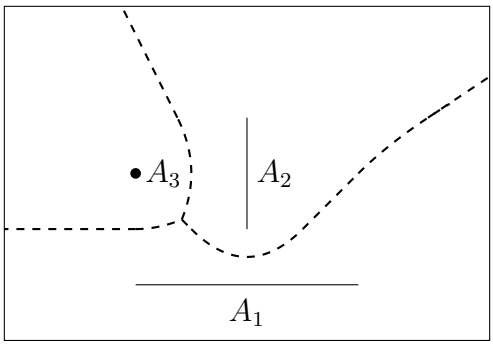


Figure 2.4: An example of the GVD in the plane for the set of objects A_1, A_2 and A_3. A notable difference to the VD, see Figure 2.2a, is that the cell-boundaries are no longer piecewise linear.

In Figure 2.4, a simple example is shown, and it is clear that GVD is very different to the VD in terms of complexity even for simple objects. The VD could be determined from half-spaces or the Voronoi vertices, but the GVD is dependent on the shape of the generator objects. Some work on computing the GVD can be found in [25, 26, 27]; and as [28] states, the conclusion is that for general objects or objects represented by many faced polyhedrons, there is a lack for exact and efficient algorithms. For example, [26] gives an $\mathcal{O}(n \log^2 n)$ algorithm where n is the total number of polyhedron faces, unfortunately n might be very large ($\approx 10^6$) in a robot station so it is impractical to consider every face separately when constructing the diagram. However, since the GVD has many areas of application, also many approximate algorithms have been developed, some of these algorithms will be considered in Section 3.4.

2.8 Generalised travelling salesperson problem

As introduced in Section 1.3 the Generalised travelling salesperson problem (GTSP) is a problem where a number of cities have been clustered into sets of cites; then a salesperson needs to find the shortest tour visiting one city in each cluster. The

algorithm suggested in Section 3.2 will partition the complete problem —recall Definition 1.5.2— generating a number of GTSPs, one for each robot in the station. To further complicate the problem, the edge cost (i.e. distance between two cities) in these GTSPs is the time of the shortest robot path between the two corresponding positions of the robot. Computing these paths is in itself a non-trivial task, since the robot may not collide with any stationary geometry; recall Figure 1.3a.

Introducing theory and algorithms to solve the GTSP is considered to be outside the scope of this project and instead the robot module in the software IPS is used to solve this problem. IPS provides both the sequence in the order of which the robot performs the tasks as well as the paths between the tasks that the robot moves along. We consider that this problem is well solved by IPS; in fact, IPS is able to well solve a slightly more difficult problem, which is the MGTSP, i.e., when considering multiple robots where some tasks can be reached by more than one robot. This more complicated problem arises in the algorithm as applied to instances with very large robot stations consisting of multiple robot cells, the workpiece is moved between the cells. Hence, even though the robots are partitioned into separate spaces, since the workpiece is moved, a task may still be performed by several alternative robots.

3
Methods

This chapter treats the suggested model, proposed algorithm, related work, and considered alternatives. Starting with presenting the model, then continuing with an overview explaining the algorithm and its components. In Chapter 4, the implementation of the algorithm is explained, while this chapter has a mathematical emphasis, and may thus seem a bit concise.

3.1 Model of the complete problem

Recall that the goal of this thesis is to give good solutions to the *complete problem*; the Definition 1.5.2 is now repeated.

Definition 3.1.1 (complete problem). Generate robot programs with minimal the cycle time, performs all tasks, has a clearance to the environment, and satisfies the robot-partitioning constraint (recall Definition 1.5.1).

There are many ways to represent this problem by a mathematical model, as mentioned in Section 1.3. The problem may be modelled as a min-max generalised multiple travelling salesperson with an additional partitioning constraint. However, even without this extra constraint this problem is not as well-studied as the min-sum multiple travelling salesperson problem or its generalisation where the cities are grouped into sets of cities and the salesperson's should visit one city in each given set of cities, see [7]. As [6] states: there is a lack of algorithms developed for this generalisation of the min-max multiple travelling salesperson problem. Our algorithm, summarised in Section 3.2, is based on the idea that given a robot-partitioning surface the decomposed problem is well solved by IPS, the focus is thus to find the best robot-partitioning surface by repeatedly solving an approximation of the complete problem. First, some notation will be established in Section 3.1.1. Then, a model of the complete problem, suitable for the algorithm, is presented in Section 3.1.2.

3.1.1 Notation

Before formulating the model, we introduce the notation used.

- The robot-robot clearance limit is denoted d_c[cm].
- $\mathcal{R}$ denotes the set of all robots.

- $\mathcal{T}$ denotes the set of all tasks to be performed.
- $\mathcal{J}_{rt}$ denotes the (possibly empty) set of joint configurations, i.e. alternative positions for robot r at task t. This set is often continuous but in such cases it is discretized.
- The (possibly empty) set of pairwise collisions $\mathcal{D}_{rtj}$ denotes the joint configurations which does not satisfy clearance limit to the joint configuration $j \in \mathcal{J}_{rt}$ and is defined as

$$\mathcal{D}_{rtj} := \left\{ \bar{j} \in \mathcal{J}_{\bar{r}\bar{t}} \colon r \neq \bar{r},\ t \neq \bar{t},\ d(R(r,t,j), R(\bar{r},\bar{t},\bar{j})) \leq d_c \right\}, \tag{3.1}$$

  where $R(r,t,j)$ denote the volume of the robot r at task t with alternative j, and d is the shortest distance between the two positioned robots.
- The variable x_{rtj} equals 1 if task t is performed using robot r and alternative $j \in \mathcal{J}_{rt}$; $x_{rtj} = 0$ otherwise. Note that this choice involves neither the task sequence nor the path between tasks. Sometimes these will be needed as a vector and hence $\mathbf{x} \in \mathbb{B}^m$ is used; $m = \sum_r \sum_t |\mathcal{J}_{rt}|$ is the total number of variables and $\mathbf{x}_l \equiv x_{rtj}$, i.e., l is the linear index of rtj.

3.1.2 The model

An exact model of the complete problem, recall Definition 3.1.1, is expressed as to

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimise}} \quad & c(\mathbf{x}), && && \text{(3.2a)}\\
\text{s.t.} \quad & \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{J}_{rt}} x_{rtj} = 1, && t \in \mathcal{T}, && \text{(3.2b)}\\
& x_{rtj} + x_{\bar{r}\bar{t}\bar{j}} \leq 1, && (\bar{r}\bar{t}\bar{j}) \in \mathcal{D}_{rtj},\ j \in \mathcal{J}_{rt},\ t \in \mathcal{T}\ , r \in \mathcal{R}, && \text{(3.2c)}\\
& x_{rtj} \in \{0,1\}, && j \in \mathcal{J}_{rt},\ t \in \mathcal{T},\ r \in \mathcal{R}, && \text{(3.2d)}
\end{aligned}$$

where $c\colon \mathbb{B}^m \to \mathbb{R}$ denotes the cost of an assignment, which is the maximum travel time among the robots. The travel time of a robot is the time of the optimal sequence for the robot to visit all tasks assigned by the values of the variables $\mathbf{x}$, using the optimal paths between tasks, while satisfying the partitioning constraint (recall Definition 1.5.1). In fact, even though the robots are assigned to different tasks they may still interact since the partitioning surface is not uniquely determined. We define $c(\mathbf{x}) := \infty$ whenever no partitioning surface exist for that particular assignment.

(3.2b) simply state that each task should be performed exactly once. (3.2c) is a necessary condition for the existence of a partitioning surface, since it ensure that the robot-clearance at the assigned task is sufficient, hence for all values of $\mathbf{x}$ for which (3.2c) is not satisfied, it holds that $c(\mathbf{x}) = \infty$. The result of including the constraints (3.2c) is that, for any feasible solution to (3.2), there exist a surface, partitioning the space such that each robot is at least a distance $\frac{d_c}{2}$ from the surface at any position given by any feasible values of $\mathbf{x}$, but not necessarily including the paths between the positions. The constraints (3.2c) is called the *conflict constraints* and will be useful when solving the problem (3.2).

3.2 Algorithm overview

The purpose of the algorithm is to give a good solution to the model of the complete problem. Recall that for any assignment $\mathbf{x}$ feasible in (3.2), there exists a surface that partitions $\mathbb{R}^3$ such that the robots are separated and, at any position given by $\mathbf{x}$, the robots clear the surface with a distance of $\frac{d_c}{2}$. If this surface is given, the complete problem separates into a set of subproblems, one for each robot, which can be well solved by IPS, recall Section 2.8. Hence, for a given assignment $\mathbf{x}$, the objective function in (3.2a), i.e., $c(\mathbf{x})$, can be evaluated, or well approximated. Here, it is assumed that the partitioning surface is well approximated by the GVD; hence the objective function is only approximated. How to find the optimal partitioning surface is discussed in Section 6.1.

However, evaluating $c(\mathbf{x})$ approximately is time consuming and hence cannot be done a large number of times. Hence, we suggest the algorithm illustrated in Figure 3.1. A main component is the approximate problem (see Section 3.3) which approximates the complete problem (3.2) by estimating the objective $c(\mathbf{x})$; this to give good candidate solutions to the complete problem. The approximate problem accounts for that the robot-robot clearance limit is satisfied during the task operation but not on the path between tasks. In other words, the entire volume swept along the robot path is not considered, but only snapshots of it at the tasks. The constraints (3.2c) need to be constructed and becomes a major pre-processing step in the algorithm; for details, see Section 4.2.

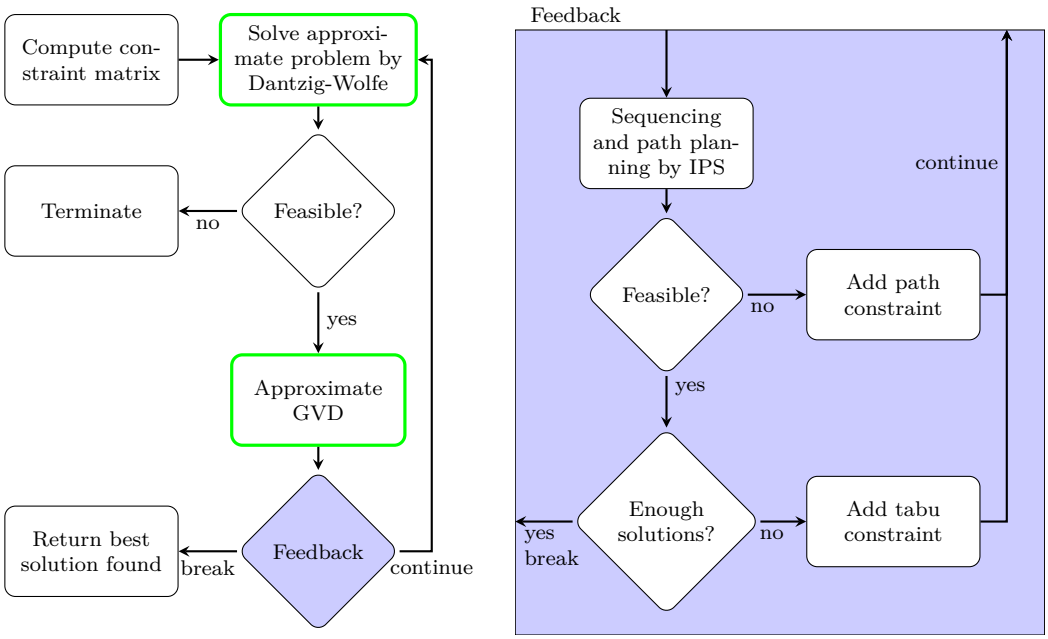


Figure 3.1: Flowchart illustrating how the parts of the algorithm connect. Components with thick green border are detailed in Sections 3.3 and 3.4. The blue feedback block is explained in Section 3.5.

Using a solution from the approximate problem, an approximation of a partitioning surface of the complete problem may be created. The robots may be positioned

at all assigned tasks simultaneously and their volumes constitute the objects creating the GVD. In Section 3.4, some algorithms for approximating the GVD are presented.

Using these two components —the approximate problem and the approximate GVD— an iterative process is started to find good candidates for the complete problem; see Section 3.5. The process accounts for two errors.

First error accounted for is the surface may block all paths to a specific task and hence no feasible solution exists in the complete problem. This issue is corrected by adding additional constraints to the approximate problem, ensuring that a path exists.

The second error accounted for is the approximation made of the complete problem. This means that the solution $\mathbf{x}$ that is optimal to the approximate problem might not be optimal to the complete problem. Nevertheless, the true optimal solution will be near-optimal in the approximate problem, since the optimal solution needs to be balanced between the robots and thus giving a good approximate solution value. Hence by iteratively solving the approximate problem, in each iteration adding a constraint preventing any solutions from reoccurring, we find many near-optimal solutions to the approximate problem, all of which are candidates for the optimal solution to the complete problem, see Section 3.5.2.

At last, if the algorithm has found a feasible solution to the complete problem it returns the best such found during the iterative process. If the algorithm fails to provide any solution, then this indicates that the complete problem (3.2) has no solution, but due to the added path-constraints there is no strict guarantee that the complete problem has no solution; see Section 3.5.1 for details.

To measure how good the solutions provided by the algorithm are in terms of robot station cycle time; they will be compared with the existing solver in IPS, in which robot collisions are handled by synchronisations messages; see Section 5.3.

3.3 Approximate problem

In order to solve the complete problem (3.2) we need to express the cost function c explicitly. The approach here suggested only requires an approximate solution as described in Section 3.2. Hence, an approximation of (3.2a) is made so that the problem may be approximately solved.

3.3.1 Semi-assignment problem with conflicts

The following approximation of (3.2a) is made:

$$c(\mathbf{x}) \approx \tilde{c}(\mathbf{x}) := \max_{r \in \mathcal{R}} \left\{ \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}_{rt}} c_{rtj} x_{rtj} \right\}. \tag{3.3}$$

This is a rough estimate but it will give surprisingly good results; see Section 5.3. The value of c_{rtj}, i.e., the cost of an assignment, is discussed in Section 4.

In our implementation it becomes crucial to reduce the number of constraints in (3.2), in order to speed up the computations. Therefore, the constraints (3.2c) are

reformulated as (3.4a) (or its vector equivalence (3.4b)). The equivalents between the constraints (3.2c) and (3.4a) is established by first, summation over the set $\mathcal{D}_{rtj}$, using the assignment constraint (3.2b) to bound the sum of x_{rtj} by $|\mathcal{T}|$. And second, by combining the constraints (3.4a) for rtj and $\bar{r}\bar{t}\bar{j}$ to retrieve the constraint (3.2c) for $(\bar{r}\bar{t}\bar{j}) \in \mathcal{D}_{rtj}$.

$$x_{rtj} + \frac{1}{|\mathcal{T}|} \sum_{(\bar{r}\bar{t}\bar{j}) \in \mathcal{D}_{rtj}} x_{\bar{r}\bar{t}\bar{j}} \leq 1, \qquad j \in \mathcal{J}_{rt},\ r \in \mathcal{R},\ t \in \mathcal{T}. \tag{3.4a}$$

$$\left(\mathbf{I}_m + \frac{1}{|\mathcal{T}|}\mathbf{D}\right)\mathbf{x} \leq \mathbf{1}_{m,1}. \tag{3.4b}$$

Here, $\mathbf{1}_{m,1}$ is the unit column vector of size m and $\mathbf{D}$ is an $m \times m$-matrix of zeros and ones: one indicates that the variables of the corresponding column, e.g. x_{rtj}, and row, e.g. $x_{\bar{r}\bar{t}\bar{j}}$, are in collision according to $\mathcal{D}_{rtj}$. Note that, since $\mathcal{D}_{rtj}$ is symmetric, which is clear from (3.1), also $\mathbf{D}$ is symmetric.

To conclude, using approximation (3.3) of the cost function and reformulating the constraints (3.2c) as (3.4b), the approximation of problem (3.2) becomes:

$$\underset{\mathbf{x}}{\text{minimise}} \quad \max_{r \in \mathcal{R}} \left\{ \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}_{rt}} c_{rtj} x_{rtj} \right\}, \tag{3.5a}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{J}_{rt}} x_{rtj} = 1, \qquad t \in \mathcal{T}, \tag{3.5b}$$

$$\left(\mathbf{I}_m + \frac{1}{|\mathcal{T}|}\mathbf{D}\right)\mathbf{x} \leq \mathbf{1}_{m,1}, \tag{3.5c}$$

$$x_{rtj} \in \{0,1\}, \quad j \in \mathcal{J}_{rt},\ r \in \mathcal{R},\ t \in \mathcal{T}. \tag{3.5d}$$

The problem (3.5) is a min-max semi-assignment problem with conflict constraints (3.5c), causing excluding conflicts among the assignments.

Relaxing the conflict constraint from the problem would lead to a min-max semi-assignment problem, which is still not a standard problem. In fact, works as [29] do not consider this type of problem. Note that the generalised assignment problem (GAP) resembles this problem: it includes a resource constraint and the objective (3.5a) may be viewed as finding the minimal level of resources, expressed as:

$$\underset{\mathbf{x},\, z}{\text{minimise}} \quad z, \tag{3.6a}$$

$$\text{s.t.} \quad \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}_{rt}} c_{rtj} x_{rtj} - z \leq 0, \qquad r \in \mathcal{R}, \tag{3.6b}$$

$$\text{(3.5b), (3.5c) \& (3.5d) hold.} \tag{3.6c}$$

This indicates that the problem (3.5) is quite hard, since the GAP is an NP-hard problem, see [30].

Motivated by this similarity to the GAP, a solution procedure to solve the GAP can be adjusted to solve this similar problem; the survey [31] suggests a relaxation of the resource constraint (3.6b); [14] suggests using this relaxation with a Dantzig-Wolfe decomposition. However, since the resource constraint (3.6b) depends on

the variable z, the approximate problem is instead decomposed by relaxing the assignment constraint (3.5b), which is the second best alternative according to [14].

There are three alternatives for using this decomposition:

- The first alternative is to relax only the assignment constraints (3.5b) in (3.6); this is described in Section 3.3.3.
- The second alternative is to include also the min-max objective from (3.5) in the subproblem; this is described in Section 3.3.4.
- The third alternative is to include both the assignment constraints (3.5b) and the conflict constraints (3.5c) in the subproblem; this is described in Section 3.3.5.

Before going into details on these three formulations, an important observation is made which will lead to a very effective way of finding a feasible solution to both the approximate problem (3.5) and the complete problem (3.2).

3.3.2 Feasibility heuristic - MiniSat

It is of interest to convert the constraints of problem (3.2) into conjunctive normal form (CNF), because there exist very efficient solvers for CNF expressions; here *MiniSat* [32] is used; see Section 4.1.3 for details. For recap on CNF, recall Section 2.1.

For feasibility of the problem (3.2), the constraints can be written in conjunctive normal form (CNF): for any assignment of $\mathbf{x}$, the constraints (3.2b) and (3.2c) are either satisfied or violated, and an equivalent CNF expression exists that evaluates to true or false, respectively.

The constraints (3.2b) are primarily a conjunction of *or*–operands, since each task should be assigned to at least one robot. This results in the expression

$$A := \bigwedge_{t\in\mathcal{T}} \bigvee_{r\in\mathcal{R}} \bigvee_{j\in\mathcal{J}_{rt}} x_{rtj}, \tag{3.7}$$

where the careful reader might object since the constraints (3.2b) states that each task should be performed exactly once. To counter this the expression

$$B := \bigwedge_{t\in\mathcal{T}} \bigoplus_{r\in\mathcal{R}} \bigoplus_{j\in\mathcal{J}_{rt}} x_{rtj} \tag{3.8}$$

may be used instead. The operator $\oplus$ is the *n-xor* operator, which can be reduced to a CNF but the number of *clauses* is rather large; recall Section 2.1.

The constraints (3.2c) are simple to reduce to a CNF expression, since we only need to restrict all members of $\mathcal{D}_{trj}$ to be false whenever x_{trj} is true, and vice versa. This is accomplished by the definition

$$C := \bigwedge_{rtj} \left(\bigwedge_{(\bar{r}\bar{t}\bar{j})\in\mathcal{D}_{rtj}} \left(\neg x_{rtj} \vee \neg x_{\bar{r}\bar{t}\bar{j}}\right)\right). \tag{3.9}$$

Hence, to determine the feasibility of (3.2) a CNF solver may be used. Moreover, the problem may be simplified by only considering A and C from (3.7) and (3.9), respectively. The solution given may then contain multiple assignments for a single task. Anyway, pruning a given solution until each task is performed exactly once is simple. Hence, the large set of clauses B in (3.8) is redundant.

3.3.3 Decomposition with simple subproblem

Following the notation introduced in Section 2.3, the problem (3.6) may be decomposed using a Dantzig-Wolfe decomposition, by introducing the set

$$\mathbf{S}^* := \left\{ \mathbf{x} \in \mathbb{B}^m \colon \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{J}_{rt}} x_{rtj} = 1,\ t \in \mathcal{T} \right\}, \tag{3.10}$$

which replaces constraints (3.5b) and (3.5d). Therefore, this set may be enumerated as $\mathbf{S}^* = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n\}$, where each $\mathbf{x}^j$ is an assignment that might contain collisions.

For a more concise expression, the constraints (3.6b) are written in matrix form. We get the equivalent condition (3.11), where the $|\mathcal{R}| \times m$ matrix $\mathbf{C}$ has one row per robot and one column per alternative position, $C_{rl} := 1$ if alternative l concerns robot r; $C_{rl} := 0$ otherwise, and the constrains are expressed as

$$\mathbf{C}\mathbf{x} - z \leq \mathbf{0}_{|\mathcal{R}|,1}. \tag{3.11}$$

Hence, the following IMP is equivalent to the approximate problem (3.6):

$$\begin{aligned}
\underset{\lambda_k,\, z}{\text{minimise}} \quad & z, && && \text{(3.12a)}\\
\text{s.t.} \quad & \sum_{1 \leq k \leq n} \mathbf{C}\mathbf{x}^k \lambda_k - z \leq \mathbf{0}_{|\mathcal{R}|,1}, && && \text{(3.12b)}\\
& \sum_{1 \leq k \leq n} (\mathbf{I}_m + \frac{1}{|\mathcal{T}|}\mathbf{D})\mathbf{x}^k \lambda_k \leq \mathbf{1}_{m,1}, && && \text{(3.12c)}\\
& \sum_{1 \leq k \leq n} \lambda_k = 1, && && \text{(3.12d)}\\
& \lambda_k \in \{0,1\}, && k = 1, \ldots, n. && \text{(3.12e)}
\end{aligned}$$

From (2.15) we get the corresponding subproblem (the z-column is treated as the slack columns in Section 2.2, i.e., since it is in the restricted master problem there is no need to find its reduced cost) to

$$\underset{\mathbf{x} \in \mathbf{S}^*}{\text{minimise}} \left\{ 0 - \bar{\boldsymbol{\gamma}}^{\mathrm{T}} \mathbf{x} - \bar{q} \right\}, \qquad \text{where} \qquad \bar{\boldsymbol{\gamma}} := \mathbf{C}^{\mathrm{T}} \bar{\boldsymbol{\mu}} + \left(\mathbf{I}_m + \frac{1}{|\mathcal{T}|}\mathbf{D} \right) \bar{\boldsymbol{\pi}}. \tag{3.13}$$

Note that the matrix of conflicts $\mathbf{D}$ is not transposed since it is symmetric. The solution of (3.13) reduces to finding the largest $\bar{\gamma}_{rtj}$ for each task t.

In Section 2.3 the lower bound (2.19) of MP is given. Considering z as a column in the RMP, it will trivially have a non-negative reduced cost in an optimal solution

to the RMP. Therefore, the bounds still hold and by replacing $\bar{z}$ (optimal value of RMP) using linear programming duality and inserting the problem of finding the least negative reduced cost $\bar{c}^*$. The lower bound for the approximate problem (3.12) takes the explicit form

$$\min_{\mathbf{x}\in\mathbf{S}} \left(-\bar{\boldsymbol{\gamma}}^{\mathrm{T}}\mathbf{x} - \bar{q}\right) + \bar{\boldsymbol{\pi}}^{\mathrm{T}}\mathbf{1}_{m,1} + \bar{q} \leq z^*. \tag{3.14}$$

Here the MP (3.12) is not decomposed over the tasks, even though it might have been since it is a linear optimization problem and we may define

$$\mathbf{S}_t^* := \left\{ \mathbf{x} \in \mathbb{B}^m \colon \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{J}_{rt}} x_{rtj} = 1,\ x_{r\bar{t}j} = 0,\ \bar{t} \in \mathcal{T} \setminus \{t\} \right\}, \tag{3.15}$$

which is clearly a partition of $\mathbf{S}^*$. However —recall the structure of the decomposition (2.17)— the convexity constraint in the MP would be identical to the relaxed constraint (3.5b). Moreover, a member of S_t^* has exactly one variable set to one; the corresponding column in the MP will thus be identical to the column in the approximate problem. These two facts give the conclusion that the original problem (3.6) is identical to the MP; the relaxed MP will thus be a continuous relaxation of the original problem.

With this observation we conclude that the bounds retrieved from the MP (3.12) will be equivalent to those of a continuous relaxation of (3.6). The two reasons for solving the MP without decomposing the subproblem are (i) that the subproblem is easily solved so there is no reason in further simplifying it, and (ii) from early tests it turns out that not many columns need to be generated and hence the size of the RMP will be smaller, and therefore faster to solve compared to the MP (3.6) with the integrality relaxed; see Section 5.1. An additional reason for not decomposing the problem is that the identical problem, the LP version of (3.6), will be solved and used as a reference, using an ILP solver.

Finally, as noted in Section 2.4, a branching procedure will be used to retrieve an integral solution to the MP; this procedure will be presented in Section 3.3.7. This since the other suggested decompositions, presented in Sections 3.3.4 and 3.3.5, respectively, will use the same branching rule.

3.3.4 Decomposition with min-max subproblem

A motivation for not using the first decomposition is because it results in an easily solved subproblem and actually somewhat harder subproblems are usually preferred, as mentioned in [14]. An example given by [14] is to include the resource constraint in the subproblem in order solve a GAP. This may seem counterintuitive but it may be viewed as the subproblem solves a greater part of the approximate problem, and this typically results in better lower bound from the MP, compared to a simple subproblem.

With this in mind the following suggestion is made to solve the approximate problem on the form (3.5) using definition of $\mathbf{S}^*$ given in (3.10). Let $\tilde{c}(\mathbf{x})$ denote the

approximation of $c(\mathbf{x})$ given in (3.3), the (approximate) MP is expressed as to

$$\underset{\lambda_k}{\text{minimise}} \quad \sum_{1\leq k\leq n} \tilde{c}(\mathbf{x}^k)\lambda_k, \tag{3.16a}$$

$$\text{s.t.} \quad \sum_{1\leq k\leq n} \left(\mathbf{I}_m + \frac{1}{|\mathcal{T}|}\mathbf{D}\right)\mathbf{x}^k\lambda_k \leq \mathbf{1}_{m,1}, \tag{3.16b}$$

$$\sum_{1\leq k\leq n} \lambda_k = 1, \tag{3.16c}$$

$$\lambda_k \in \{0,1\}, \qquad k = 1,\ldots,n. \tag{3.16d}$$

Hence, the subproblem (2.18) can then be expressed as to

$$\underset{\mathbf{x},\, z}{\text{minimise}} \quad -\bar{\boldsymbol{\gamma}}^{\mathrm{T}}\mathbf{x} + z - \bar{q}, \tag{3.17a}$$

$$\text{s.t.} \quad \mathbf{C}\mathbf{x} - z \leq \mathbf{0}_{|\mathcal{R}|,1}, \tag{3.17b}$$

$$\mathbf{x} \in \mathbf{S}^*, \tag{3.17c}$$

where $\bar{\boldsymbol{\gamma}} := (\mathbf{I}_m + \frac{1}{|\mathcal{T}|}\mathbf{D})\bar{\boldsymbol{\pi}}$ and $\tilde{c}(\mathbf{x})$ from (3.16a) has been replaced by introducing z and constraints (3.17b). This subproblem again resembles the GAP but here the resource should be minimised as well. As mentioned in [31] a viable approach is to employ a Lagrangian relaxation of constraints in the definition of the set S^* in (3.17c), but for this case that would yield a set of knapsack problems where also the resource z should be minimised and no literature for solving such a problem was found. Hence, here the Lagrangian relaxation of the resource constraint (3.17b) is suggested.

First, we reformulated the problem by realising that in an optimal solution to (3.17) at least one of the constraints (3.17b) will be active, we call it $\tilde{r}$. If $\tilde{r}$ would be know, the problem to

$$\underset{\mathbf{x}}{\text{minimise}} \quad \mathbf{C}_{\tilde{r}}^{\mathrm{T}}\mathbf{x}_{\tilde{r}} - \bar{\boldsymbol{\gamma}}^{\mathrm{T}}\mathbf{x} - \bar{q}, \tag{3.18a}$$

$$\text{s.t.} \quad -\mathbf{C}_{\tilde{r}}^{\mathrm{T}}\mathbf{x}_{\tilde{r}} + \mathbf{C}_r^{\mathrm{T}}\mathbf{x}_r \leq 0, \qquad r \in \mathcal{R}\setminus\{\tilde{r}\}, \tag{3.18b}$$

$$\mathbf{x} \in \mathbf{S}^* \tag{3.18c}$$

is equivalent to (3.17), where robot, $\tilde{r}$, is assumed to possess the longest approximate cycle time. By solving (3.18) for every possible $\tilde{r}$ the best solution over these cases is also optimal in (3.17).

Now, since the main goal of solving the subproblem is not to find an optimal solution but only to find a negative reduced cost for the MP, this allows us to solve the problem to near-optimality; as noted in [33], a Lagrangian heuristic is applicable in such a case. With this motivation, a Lagrangian relaxation with respect to the constraints (3.18b) is made for each subproblem, yielding the problem to find the optimum of the Lagrangian dual, given by

$$\max_{\mu_r\geq 0}\left[\min_{\mathbf{x}\in\mathbf{S}^*}\mathbf{C}_{\tilde{r}}\left(1-\sum_{r\in\mathcal{R}\setminus\{\tilde{r}\}}\mu_r\right)\mathbf{x}_{\tilde{r}} + \sum_{r\in\mathcal{R}\setminus\{\tilde{r}\}}\mu_r\mathbf{C}_r\mathbf{x}_r - \bar{\boldsymbol{\gamma}}_r^{\mathrm{T}}\mathbf{x}\right] - \bar{q}. \tag{3.19}$$

If we define $\mu_{\tilde{r}} := 1 - \sum_{r\in\mathcal{R}\setminus\{\tilde{r}\}}\mu_r$, (3.19) reduces to

$$\max_{\mu_r\geq 0}\left[\min_{\mathbf{x}\in\mathbf{S}^*}\left(\bar{\boldsymbol{\mu}}^{\mathrm{T}}\mathbf{C} - \bar{\boldsymbol{\gamma}}^{\mathrm{T}}\right)\mathbf{x}\right] - \bar{q}, \tag{3.20}$$

which is a simple problem to solve since for any given $\boldsymbol{\mu}$, the inner minimisation can be evaluated in $\mathcal{O}(m)$ time and since the number of robots is usually rather small, $\boldsymbol{\mu}$ is in a low dimensional space. To solve this, a subgradient optimization method may be used (for details on subgradient optimisation, see [34, Ch. 6.4] and Section 4.3.3). Note that there is no imminent need for a primal feasibility heuristic since any $\mathbf{x} \in \mathbf{S}^*$ is also feasible in (3.17).

3.3.4.1 Lower bound for min-max subproblem

A lower bound for the branching procedure needs to be determined and as given in (2.19) it relies on the optimal solution of the subproblem. However, since only a near optimal solution is achieved, a lower bound to this, easily retrieved from the Lagrangian dual (3.20), may be used instead. However, since the Lagrangian dual involves solving an integer problem, there is a duality gap of unknown size, which may destroy the desired property of an improved lower bound.

3.3.5 Decomposition with conflicted subproblem

As said, there might be great gain in decomposing using a harder subproblem; this since it may give better bounds. In this particular case we may restrict the set $\mathbf{S}^*$ from (3.10) to only include columns satisfying the conflict constraint (3.5c), that is

$$\mathbf{SC}^* := \{\mathbf{x} \in \mathbb{B}^m \colon \text{(3.5b) \& (3.5c) hold}\}. \tag{3.21}$$

The IMP thus becomes to

$$\begin{aligned}
\underset{\lambda_k,\, z}{\text{minimise}} \quad & z, && && \text{(3.22a)}\\
\text{s.t.} \quad & \sum_{1\le k\le n} \mathbf{C}\mathbf{x}^k\lambda_k - z \le \mathbf{0}_{|\mathcal{R}|,1}, && && \text{(3.22b)}\\
& \sum_{1\le k\le n} \lambda_k = 1, && && \text{(3.22c)}\\
& \lambda_k \in \{0,1\}, && k = 1,\dots,n. && \text{(3.22d)}
\end{aligned}$$

This is a rather small problem; note that the number of constraints is only one more than the number of robots in the station. Thus, any basic feasible solution to the corresponding MP will be a combination of only that few columns.

The corresponding subproblem to the MP becomes to

$$\underset{\mathbf{x}\in\mathbf{SC}^*}{\text{minimise}} \ \left\{0 - \bar{\boldsymbol{\gamma}}^{\mathrm{T}}\mathbf{x} - \bar{q}\right\}, \qquad \text{where} \qquad \bar{\boldsymbol{\gamma}} := \mathbf{C}^{\mathrm{T}}\bar{\boldsymbol{\mu}}. \tag{3.23}$$

Here, $\bar{\boldsymbol{\mu}}$ are the optimal values of the dual variables corresponding to the constraints (3.22b) in the RMP and $\bar{q}$ is again the optimal values of the dual variable corresponding to the convexity constraint (3.22d).

As in Section 3.3.3, the lower bound is derived from (2.19) and takes the explicit form

$$\min_{\mathbf{x}\in\mathbf{S}} \left\{-\bar{\boldsymbol{\gamma}}^{\mathrm{T}}\mathbf{x} - \bar{q}\right\} + \bar{q} \le z^*. \tag{3.24}$$

3.3.5.1 Solving the conflicted subproblem

The approach suggested to solve the subproblem (3.23) is to relax the conflict constraint, yielding the simple subproblem (3.13), and then resolving any resulting conflicts by a branching procedure.

Thus, the subproblem is solved by a branching procedure, which works as follows. In each node, the subproblem (3.23) is relaxed by considering the larger set $\mathbf{S}^*$ instead of $\mathbf{SC}^*$. If the solution $\bar{\mathbf{x}}$ to this relaxed problem is feasible in the subproblem, i.e., if it contains no conflicts, then it is optimal in the current branch.

If the solution contains conflicts, then two new branches are made. On the left branch the variable with the most conflicts are set to 1 and all its conflicted variables are removed (i.e., set to 0), and on the right branch the variable with most conflicts is removed (i.e., set to 0). Note that, in each of these branches at least one variable conflict is removed; hence, when sufficiently many such constraints have been imposed on the set $\mathbf{S}^*$, the resulting solution fulfils $\bar{\mathbf{x}} \in \mathbf{SC}^*$.

Note also that the relaxed problem offers a lower bound on the optimal value of the subproblem and hence whenever a node has an optimal value to the relaxed subproblem that exceeds that of the best obtained subproblem solution, then this branch can be cutoff the tree.

This approach of solving the conflicted subproblem is heavily dependent on a preprocessing step (discussed in Section 4.3.4) that reduces the number of variables.

3.3.6 Several optimal solutions to MP

When an optimal solution to the MP of the approximate problem (3.5) is found, it is not likely a unique optimal solution since the objective (3.5a) only minimises the longest completion time and there might exist equally costly alternatives. Since the goal is to retrieve a solution to the IMP of the approximate problem (3.5) and since the method to reach there is a branching procedure (see Section 3.3.7) it might be possible to find another optimal solution to the MP containing fewer fractional variables.

Before branching, the solution λ^* is first converted to the original variables $\mathbf{x}^*$, according to (2.11). The solution is called fractional if any element of $\mathbf{x}^*$ is non-integer, i.e., if a task is partially performed by each of several alternatives. The branching rule aims to prevent fractional solutions, however as noted there might exist other LP optimal solutions with fewer fractional tasks.

Here, a simple approach is applied to an LP optimal solution to the approximate problem (3.5) resulting in fewer fractional tasks: For every alternative that is fractional, i.e., for each variable value $x^*_{rtj} \in (0, 1)$, set it to 1 and all other alternatives corresponding to the same task to 0, if the resulting solution is still feasible and optimal the current changes are kept, otherwise the changes are discarded. Note that even though the aim is to reduce the number of fractional variable values so that the branching will be more efficient, this approach might also result in an integral solution, so that further branching is not needed in the current node.

3.3.7 Branching rule

Any solution to the MP will in particular satisfy the assignment constraint (3.2b). Hence, if one variable $x^*_{rtj} \in (0, 1)$, then at least one more variable is also fractional. Using this observation, a branching rule inspired by [35] is formulated.

The idea is to create a balanced tree, i.e., the number of new conditions on each the two branches should be approximately equal; see [13]. One error that might occur in a fractional solution is that a task is partly performed by different robots. In this case the left branch will only allow the robot with the highest approximate cycle time to perform the task and the right branch will not allow this robot to perform the task. If this is not the case, then it must be the case that a task is performed by a single robot using two or more alternative positions. An inspection of the approximate problem (3.5) reveals that this can happen in an optimal solution only in the case of conflicts (3.5c) or if several integral solutions are equally good (which is partly prevented in Section 3.3.6). To prevent the later error, the branching rule aims to remove as many pairwise conflicts as possible, as alternatively stated in (3.2c).

In the latter case, the branching rule in Algorithm 3.1 is applied. The algorithm tries to balance the nodes by forcing about half of the alternatives to be used in either branch. In addition, the alternatives on either branch are grouped according to common conflicts. This since if all alternatives allowed for a task, have some conflict variables in common then all of these will be set to zero by the conflict constraint (3.2c), since one of them always appear in the assignment constraint (3.2b).

Using this, the tree will be searched more effectively since more alternatives are tested in each node, as explained in [35]. In addition, since the variables are grouped according to conflicts, this rule will increase the possibility to remove additional variables (see Section 4.3.1). It remains to choose the order, in which the tree created by the left and right branches is searched. Several approaches exist (see [36]) and we chose a depth first search (see Section 4.3.6).

3.4 Approximation of the GVD

For an assignment $\mathbf{x}^*$ from the approximate problem (3.2), consider the objects $R_1, R_2, \ldots, R_{|\mathcal{R}|}$, where R_r is defined as

$$R_r := \bigcup_{t \in \mathcal{T}} \bigcup_{j \in \mathcal{J}_{rt} \colon x^*_{rtj}=1} R(r, t, j), \tag{3.25}$$

where $R(r, t, j)$ denotes the volume of the robot r at task t with alternative j. Now, since the conflict constraint (3.2c) is satisfied, there is a guaranteed minimum clearance between the volumes covered by different robots; as examined in Section 2.7, these objects can be separated using the GVD concept. An exact implementation of the GVD concept would yield one Voronoi cell for each robot; in this Section we consider some approximations of these cells. In Section 3.5 the resulting approximate cells will be used to solve by which paths the assigned tasks should be performed and suggest how the solution can be improved.

Algorithm 3.1 Second branching rule

1: Find t, the fractional task with most pairwise conflicts.
2: Denote $\mathcal{A}_t := \bigcup_r \mathcal{J}_{rt} \cap B$ to be all alternatives for the task, where B denotes all alternatives allowed in the node.
3: Find r & j with highest value of x_{rtj} and $(rtj) \in \mathcal{A}_t$.
4: Let $\mathcal{V}_R := \{(rtj)\}$ and $\mathcal{C}_R := \mathcal{D}_{rtj} \cap B$
5: Find r & j with least intersection $|\mathcal{C}_R \cap \mathcal{D}_{rtj}|$, in case of ties pick the largest $|\mathcal{D}_{rtj} \cap B|$.
6: Let $\mathcal{V}_L := \{(rtj)\}$ and $\mathcal{C}_L := \mathcal{D}_{rtj} \cap B$
7: **for all** $(rtj) \in \mathcal{A}_t \setminus (\mathcal{V}_L \cup \mathcal{V}_R)$ **do**
8: **if** $|\mathcal{C}_L \cap \mathcal{D}_{rtj}| > |\mathcal{C}_R \cap \mathcal{D}_{rtj}|$ **then**
9: Update $\mathcal{V}_L := \mathcal{V}_L \cup (rtj)$ and $\mathcal{C}_L := \mathcal{C}_L \cap \mathcal{D}_{rtj}$.
10: **else if** $|\mathcal{C}_L \cap \mathcal{D}_{rtj}| < |\mathcal{C}_R \cap \mathcal{D}_{rtj}|$ **then**
11: Update $\mathcal{V}_R := \mathcal{V}_R \cup (rtj)$ and $\mathcal{C}_R := \mathcal{C}_R \cap \mathcal{D}_{rtj}$.
12: **else if** $|\mathcal{V}_L| < |\mathcal{V}_R|$ **then**
13: Update $\mathcal{V}_L := \mathcal{V}_L \cup (rtj)$ and $\mathcal{C}_L := \mathcal{C}_L \cap \mathcal{D}_{rtj}$.
14: **else**
15: Update $\mathcal{V}_R := \mathcal{V}_R \cup (rtj)$ and $\mathcal{C}_R := \mathcal{C}_R \cap \mathcal{D}_{rtj}$.
16: **end if**
17: **end for**
18: Left branch force all x_{rtj} in $\mathcal{V}_R$ to zero and thus also all x_{rtj} in $\mathcal{C}_L$ to zero. The right branch does the equivalent using $\mathcal{V}_L$ and $\mathcal{C}_R$.

There are many different approximations of the GVD; see [37, 28, 19, 38]. In this project two known algorithms are implemented and an improvement of one of them is suggested. The first algorithm ([19, p. 287]) relies on a discretization of the continuous objects and finds the Voronoi diagram for the discretization; see Section 3.4.1. The second algorithm ([28]) relies on interpolations in a distance field; see Section 3.4.2; the suggested improvement is an alternative interpolation in the same distance field; see Section 3.4.3.

3.4.1 Approximation by points

This approximation is motivated by from the fact that the Voronoi diagram can be easily computed on a set of points. Hence, by approximating the objects R_r (defined in (3.25)) using a large number of points, i.e., a discretization of the continuous objects, the Voronoi faces shared between points belonging to different objects will approximate the GVD; for a pseudocode, see Algorithm 3.2.

Algorithm 3.2 Approximation by points [19, p. 287]

1: Approximate the boundary of each object R_r with a finite point set P_r.
2: Compute the Voronoi Diagram, $\mathcal{V}$ for all points, $P = \bigcup_{r=1}^{|\mathcal{R}|} P_r$.
3: For each Voronoi cell: delete from $\mathcal{V}$ all facets shared by generator points from same point set.
4: Return $\mathcal{V}$.

In step 2 of Algorithm 3.2 the Voronoi diagram is computed for a large set of points. This is a well-studied problem and there exist many efficient algorithms for computing the Voronoi diagram in three and higher dimensions. The most popular is a lifting transformation, i.e., each point $\mathbf{p}_i \in P$ is appended the square of its norm, such that the transformed point is defined as $\mathbf{p}_i^* = [\mathbf{p}_i^{\mathrm{T}}, \|\mathbf{p}_i\|^2]^{\mathrm{T}}$. Note that $\mathbf{p}_i^* \in \mathbb{R}^{d+1}$. The convex hull of the transformed points is a Delaunay pretessellation of the point set P from which the Voronoi diagram is easily retrieved (see [19, p. 278]). To conclude, many efficient algorithms for computing the Voronoi diagram uses an efficient convex hull algorithm; an example is [39], which is in the Matlab kernel.

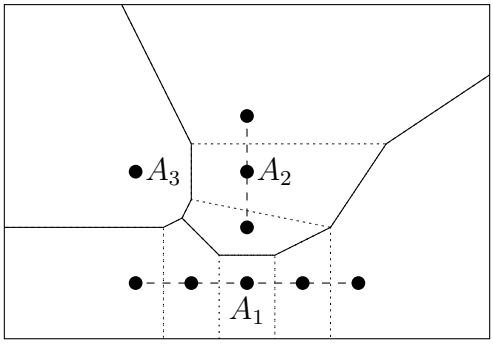


Figure 3.2: Illustrate the output of Algorithm 3.2 as applied to objects A_1, A_2 and A_3 from Figure 2.4 (in which also the exact GVD is illustrated). The nine bullet points represent the discretization, and the solid lines comprise the approximated GVD, and the dotted lines indicate the facets that are deleted in step 3.

We have used an implementation of the algorithm in [40], which utilises the fact that a Voronoi cell is an intersection of half-spaces (recall (2.22)) and a procedure (described in [41, pp. 82–83]) to reduce the number of redundant half-spaces that are tested. This procedure is described in Section 4.1.5.

Figure 3.2 shows an example of the output of Algorithm 3.2. We can conclude that both number of points and the position of the points will impact the quality of the approximation of the Voronoi cells. In Section 5.2 we investigate how many randomly distributed points are required for a good enough approximation of the GVD.

3.4.2 Approximation by distance field

This approximation algorithm of the GVD is originally suggested in [28] and it is based on the concept of a distance field. A distance field is a set of points where each point has an associated shortest distance to each of the robots R_r. The points in the distance field may be uniformly spread (see [42]) but as suggested in [28], this assumption is dropped.

We present a modified version of the algorithm in [28]. The algorithm is composed by four major steps:

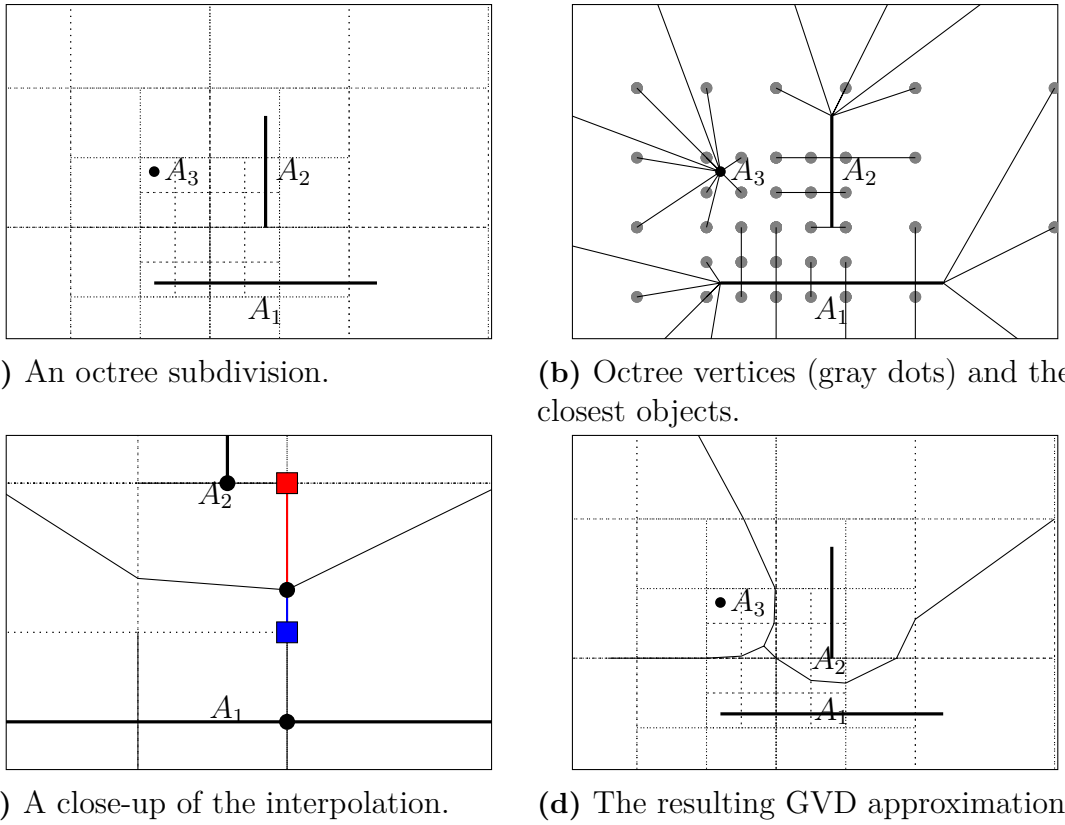
1. Determine the locations of the points in the distance field (Figure 3.3a), using the concept of octree; see Section 3.4.2.1.

2. Measure the distances to the objects in these points (Figure 3.3b); see Section 3.4.2.2.

3. Resolve ambiguities in topology by introducing additional points in the distance field; see Section 3.4.2.3.

4. Approximate points on the GVD by interpolating in the distance field (Figure 3.3d) and build GVD approximation from these points (Figure 3.3c); see Section 3.4.2.4.

In Figure 3.3, the 2D-example of a GVD in Figure 2.4 is continued using this algorithm. A 3D-example in 3D is given in Section 4.4.2.



(a) An octree subdivision.

(b) Octree vertices (gray dots) and their closest objects.

(c) A close-up of the interpolation.

(d) The resulting GVD approximation.

Figure 3.3: Illustration of the construction of the approximate GVD using distance fields, for in the example in Figure 2.4.

The main difference to the algorithm in [28] is that that we do *not* use the distance transform (which approximates the objects by a set of points, from which all distances are computed and which allows for very fast computation). The reasons are the following:

- To increase accuracy of the approximation. Since the complete algorithm consists of many parts, inaccuracy in one part may lead to erroneous conclusions in other parts.

- The use of the distance transform presented in [28] requires an extended interface to IPS (described in Section 4.1.2).

Using the distance transform in [28] is considered as further work, see Section 6.1.

3.4.2.1 Construction of distance field using octree

The suggested algorithm is based on the concept of an octree, which is a recursive subdivision of a 3D-cell into smaller cells; see Figure 3.3a for an example in $\mathbb{R}^2$; in 2D an octree is called a quadtree. Starting with a single 3D-cell representing the domain of interest, the cell is recursively subdivided until a stopping criterion is met. Each cell contains references to its parent cell and all its child cells; and as in [28] all vertices of the cell contains references to its neighbouring vertices (which might be contained in other cells). This structure allows for finding a face in a neighbouring cell in $\mathcal{O}(1)$ operations. Note that there exist many different representations of the octree, of which some other also possess this property; see [43].

The subdivision routine in Algorithm 3.3 consist two subdivision rules, the first rule ensures that any leaf in the octree intersects at most one object, and the second rule guarantees that there exist at least one empty cell separating the objects.

Algorithm 3.3 Subdivide cells [28, p.301]

1: Initiate a tree $\mathcal{T}$ and a list of new leafs $\mathcal{L} = \{c\}$, where c denotes a cell containing the whole domain.
2: **for all** $c \in \mathcal{L}$ **do**
3: **if** $|\mathcal{I}| > 1$, where $\mathcal{I} := \{r \in \{1, \dots, |\mathcal{R}|\} : c \cap R_r \neq \emptyset\}$ **then**
4: Subdivide c and push the new eight cells to the end of $\mathcal{L}$.
5: **else if** $c \cap R_r \neq \emptyset$ and $\tilde{c} \cap R_{\tilde{r}} \neq \emptyset$ for $r \neq \tilde{r}$ and $\tilde{c}$ face neighbour of c. **then**
6: Subdivide c and push the new eight cells to the end of $\mathcal{L}$.
7: **end if**
8: Remove c from $\mathcal{L}$ and insert it into $\mathcal{T}$.
9: **end for**

3.4.2.2 Measuring distance field

[28] suggests that the distance field is approximated by a distance transform, but as stated in Section 3.4.2 this is discarded. IPS has a fast routine for computing the distance to an object, but since the number of vertices (corners of octree cells) is large, Algorithm 3.4 is used to reduce the number of redundant distance calculations.

Algorithm 3.4 calculate bounds on the distance to reuse information from previous distance calculations. The idea is to avoid computing the distance to objects in remote Voronoi cells. An important reason being that this procedure is easy parallelize, measuring several vertices before updating. Figure 3.3b illustrates a 2D-example of objects and their corresponding closest vertices.

3.4.2.3 Resolve ambiguities

As noted in [28] the fact that the distance field is built in a discrete space might introduce ambiguities in the topology of the approximated GVD, i.e., several topologies

Algorithm 3.4 Determine the closest objects in the distance field

Given a set of vertices $\mathcal{V}$ in the octree. Initialise $d^{vr}_{\texttt{lb}} := 0$ and $d^{vr}_{\texttt{ub}} := \infty$ for all vertices $v \in \mathcal{V}$ and objects $r \in \{1, \ldots, |\mathcal{R}|\}$.
for all $v \in \mathcal{V}$ **do**
    let $\bar{r} := \text{argmin}_r\{d^{vr}_{\texttt{ub}}\}$
    if $\exists r \neq \bar{r}$: $d^{v\bar{r}}_{\texttt{ub}} > d^{vr}_{\texttt{lb}}$ **then**
        Measure and update $d^{vr}_{\texttt{lb}} := d^{vr}_{\texttt{ub}} := d(v, r)$ for all r where $d^{vr}_{\texttt{lb}} < d^{v\bar{r}}_{\texttt{ub}}$
        for all $\bar{v} \in \mathcal{V}$ **do**
            Update $d^{\bar{v}r}_{lb} := \max\{d^{\bar{v}r}_{lb}, d^{vr}_{lb} - d(v, \bar{v})\}$
            and $d^{\bar{v}r}_{ub} := \min\{d^{\bar{v}r}_{ub}, d^{vr}_{ub} + d(v, \bar{v})\}$.
        end for
    end if
end for

match the ambiguous cell; see Figures 3.4a and 3.4b. In the case of an ambiguous cell there may exist several ways to resolve the ambiguity; see [44].

A first attempt to resolve these ambiguities is by further subdividing the cell, hoping that the new leaf cell in the octree will be unambiguous. According to [28], a cell is ambiguous if and only if it is reduced to a simplex of dimension ≤ 3 when all neighbouring vertices with the same closest object are merged; see Figure 3.4c. This subdivision continues recursively until a maximum recursion depth is reached. Then if there are still ambiguous cells, these are resolved by adding a central hub; see Section 3.4.2.5 for details.

3.4.2.4 Ensuring better interpolation

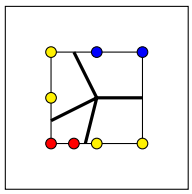
A final subdivision rule is here introduced to improve the interpolation in Section 3.4.2.5.

Recall that each octree leaf is ensured to have at least one empty buffer cell between the robots (see Section 3.4.2.1), which ensures that the GVD is enclosed in any such cell. Hence, all cells are recursively subdivided until no cell simultaneously intersects the GVD and an object.

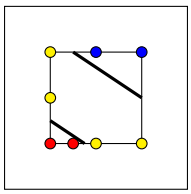
Unfortunately, to check whether a cell and the GVD intersect is in general hard, but we will use a somewhat weaker but simple check: if a cell contains vertices corresponding to different closest objects then it will contain the approximated GVD. The resulting approximate GVD is enclosed by empty buffer cells; in particular, the vertex-object distance is non-zero for each vertex used in the interpolation formula (3.29).

3.4.2.5 Compute GVD approximation

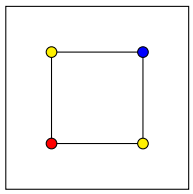
The approach suggested in [28] to compute the GVD approximation is to approximate the GVD in each cell separately, while ensuring that the interface between cells is defined unambiguously. The output for each cell will be a set of triangles, each of which represents the interface between two generalised Voronoi cells. These



(a) A topology a with central hub.



(b) A topology without a central hub.



(c) The cell with collapsed identical labels.

Figure 3.4: Illustration of an ambiguous cell and two of the possible topologies; the vertices are grouped by colours representing the closest object. (c) The cell does not become a simplex by collapsing vertices with identical labels. This example is found in [28].

triangles are computed as follows.

The first extreme point of such a triangle will be on an edge of the cell, between two adjacent vertices with different closest object. Denote the vertices v_1 and v_2 and the corresponding objects R_1 and R_2. We want to approximate the point on the line between v_1 and v_2, and which is equally close to R_1 and R_2, i.e., a point on the GVD. This is done by assuming that the distance between the respective objects and vertices change linearly along the line mathematically expressed as

$$p = v_1 + t(v_2 - v_1), \tag{3.26}$$

$$d(p, R_1) \approx d(v_1, R_1) + t[d(v_2, R_1) - d(v_1, R_1)], \tag{3.27}$$

$$d(p, R_2) \approx d(v_1, R_2) + t[d(v_2, R_2) - d(v_1, R_2)], \tag{3.28}$$

and illustrated in Figure 3.5. The solution of this system of equations 3.26 is given by

$$p = \frac{\Delta_1 v_2 + \Delta_2 v_1}{\Delta_1 + \Delta_2}, \qquad \text{where} \qquad \Delta_i := |d(v_i, R_2) - d(v_i, R_1)|. \tag{3.29}$$

Note that the values of the distance functions are not yet known but approximated by the bounds of the distance field. Therefore, the distance field is remeasured for all vertices that were the bounds do not coincide.

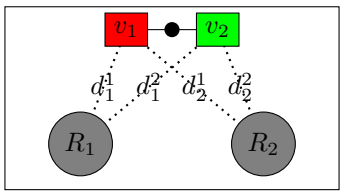


Figure 3.5: Illustration of the distances used in the interpolation (3.26), where $d_j^i := d(v_i, R_j)$.

Our linear approximation is a generalisation of the suggestion in [28], that the point on the object corresponding to the shortest distance to the vertices remains the same and which only requires a measured distance to the closest object of each vertex.

The main reason for diverting from the suggestion in [28] is, however, due to implementation reasons in the current IPS interface; see Section 4.1.2.

The second extreme point of each these triangles is located on a face of the cell. Each face of the cell is composed by 2D-cells since the neighbouring cell sharing this face may have been subdivided, see Figure 3.6. In each such 2D-cell every interpolated point (first extreme point) is connected to the mean point of all interpolated points in the 2D-cell, which becomes the second extreme point.

The last extreme point of each triangle equals the mean of all second corners in the entire cell. Note that this implies that ambiguity of a cell is resolved by use of a central hub, as illustrated in Figure 3.4a. An illustration of these triangles are given is Figure 3.6.

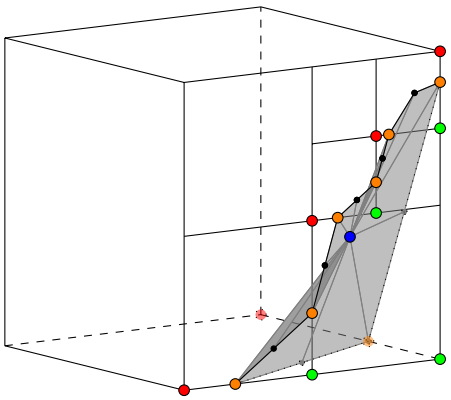
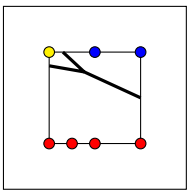


Figure 3.6: A 3D-cell of the octree. A red (green) point denotes that the corresponding vertex is closest to the red (green) object, an orange point denotes the interpolation (3.29) of a red and a green point, a black dot denotes the second corner of a triangle, and the blue point denotes the common third corner. The approximate GVD is represented by the grey triangles.

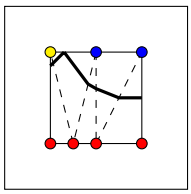
3.4.3 Distance field approximation improvement

We next suggest an improvement of the distance field approximation. The drawback of the GVD approximation described in Section 3.4.2.5 is that the two extra points introduced to create the triangles are not necessarily on or even close to the GVD. This is illustrated in Figure 3.7a: since the mean does not account for the measurements, the added point may be far from the true GVD. To remove these artefacts the triangles extreme points of the approximating the GVD will instead be located at interpolated points. To achieve this, a Delaunay tetrahedralization is made on the distance field vertices, as described in Section 2.6. Applying Property 2.6.1 (the empty circum-sphere property) yields that every Delaunay tetrahedron will be contained in a cell. Hence the triangulation may be done in parallel, considering each cell separately. All degenerate Voronoi cells are resolved in a deterministic manner, to ensure that the interfaces between cells are unambiguous.

In the given tessellation, for each tetrahedron with vertices having different closest objects, triangles representing the interface between generalised Voronoi cells need to be computed. This is, however, a simple as compared to the case of a 3D-cell and no extra points need to be introduced.



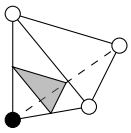
(a) Using a central hub.



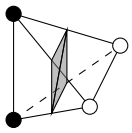
(b) Using a Delaunay tessellation.

Figure 3.7: An case for which the algorithm suggested in [28] performs poorly. (a) The yellow vertex is very close to the GVD, implying that a good approximation of the GVD should also be close to the GVD. (b) The result of the suggested improvement.

The surface that separates one object from all other objects can intersect the tetrahedron in either of just two ways. Either one or two adjacent vertices are isolated on one side of the surface –all other combinations are variants of these two; see Figure 3.8. For one isolated vertex only a triangle is needed; see Figure 3.8a. For two vertices isolated on one side of the surface two triangles are needed, and which are created by first finding two interpolated points not sharing a face in the tetrahedron. These two points will create the common edge of the two triangles, which then include one of the remaining points each; see Figure 3.8b. By repeating



(a) One vertice is isolated on one side of the surface.



(b) Two adjacent vertices are isolated on one side of the surface.

Figure 3.8: The two cases of how the tetrahedron vertices may separated by a surface, all other cases can be retrieved by rotating the vertex order and/or flipping the meaning of black and white.

this procedure of separating one object from all other objects a surface that representing the GVD. A 2D-example of the result is shown in Figure 3.7b. To see the improvement in the 3D-case see Section 4.4. The improvement made in the 3D-case is illustrated in Section 4.4.

3.5 Path planning and updating

As mentioned in Section 3.2, the GVD surface created using an optimal solution to the approximate problem (3.5) might not be optimal or even feasible. In fact, the cost function (3.3) only estimates the true cost in the problem (3.2) and since the approximate GVD is created from the robot objects positioned at the tasks without considering the paths between the tasks.

We have used IPS to solve the sequence and path-planning problems where the robots are restricted by a given separating surface. We assume that if there exists

a feasible solution for a specific robot, i.e., there exist a cycle among all assigned tasks, IPS will find it, and that the solution will be near optimal.

Here an iterative method is suggested which searches for feasible and good solutions. This search will iteratively add constraints to the approximate problem (3.5). The second phase of the search will try to balance the load between the robots, based on the cycle times of the solutions found.

3.5.1 Search for a feasible solution

If IPS is unable to find a feasible solution, then there exists at least one task for which no collision-free path from the home position can be found. To ensure that a path will exist in the next iteration, a path from the corresponding robot's home position is planned, disregarding the separating surface. Using this path, the set $\mathcal{D}_{rtj}$ of pairwise collisions defined in (3.1) is expanded: the robot object $R(r, t, j)$ will include snapshots of the robot traversing the path from its home position to the configuration $j \in \mathcal{J}_{rt}$. These snapshots are sufficiently many to make a good enough approximation of the swept volume of the robot.

Note than when this additional constraint is added, the approximate problem (3.5) may become infeasible even though a finite solution exists for the complete problem (3.2). This since the additional constraints ensures a robot a particular path to a task and hence prohibit any other robot to enter that space, hence may cause another robot to be unable to reach an assigned task. However, the additional constraint ensures a particular path that may block other tasks to be performed by any robot, but another, perhaps longer path would not cause this blockage.

3.5.2 Prohibited search for an improved solution

All assignments and corresponding surfaces found, which allow IPS to find a feasible solution to the complete problem (3.2), are stored along with their corresponding cycle times. On way to exclude an assignment $\mathbf{x}^i$ from the approximate problem (3.5) is to add the constraint

$$\left(\mathbf{x}^i\right)^{\mathrm{T}} \mathbf{x} \leq |\mathcal{T}| - 1. \tag{3.30}$$

By iteratively adding these constraints we find all near-optimal solutions to the approximate problem in increasing order, i.e., starting with the optimal, then the second best, and so on.

The optimal solution to the complete problem is probably well balanced in terms of number of tasks assigned to each robot, since a relatively large amount of time is spent performing the tasks. Hence, the optimal solution will be a near-optimal solution to the approximate problem. So by letting the number k of solutions to the approximate problem be sufficiently large, one of them will probably correspond to the optimal solution. Here, k will be heuristically determined as, e.g., the number of solution found within a given time period. However, a sufficiently large value of k can be determined by considering another approximate problem; see Section 6.1. These assignments might, however, be very similar, and since the partitioning

surface depends mainly on the robot-task assignment, i.e., not on the particular alternatives used, instead the constraint

$$\sum_{(r,t)\in S^i} \sum_{j\in\mathcal{J}_{rt}} x_{rtj} \leq |\mathcal{T}| - 1, \text{ where } S^i := \left\{(r,t) \in \mathcal{R}\times\mathcal{T} |\ \exists j \in \mathcal{J}_{rt} \colon x^i_{rtj} = 1\right\} \tag{3.31}$$

is used. Hence, the process will not find all near-optimal solutions but only all near-optimal solutions that are substantially different. Using this procedure, the solutions from the approximate problem (3.5) will eventually provide very poor solutions to the complete problem and then the process is terminated.

An additional motivation for the use of the more restrictive constraint (3.31) is that during the sequencing and path planning in IPS, the particular alternative of a task is not necessarily used —the alternatives are only required to create the GVD. Thus, this process can find an optimal solution to the complete problem (3.2) by also modifying the partitioning surface; this will be discussed in Section 6.1.

The careful reader might note that the constraint (3.31) will add to the MPs (3.12), (3.16) and (3.22). Luckily, the only difference to their respective subproblems (3.13), (3.17) and (3.23) is the definition of $\bar{\boldsymbol{\gamma}}$. In either subproblem, we get $\bar{\boldsymbol{\gamma}} := \bar{\boldsymbol{\gamma}} + \bar{\boldsymbol{\nu}}\mathbf{B}$, where $\bar{\boldsymbol{\nu}}$ are the optimal values of the LP dual variables corresponding to the new constraints and the rows in the matrix $\mathbf{B}$ correspond to the constraint (3.31). The lower bound is also modified such that in both subproblems $z_{\text{lb}} := z_{\text{lb}} + (|\mathcal{T}| - 1)\mathbf{1}^{\text{T}}\bar{\boldsymbol{\nu}}$.

A more substantial change occurs in the feasibility heuristic described in Section 3.3.2, because the additional constraint (3.31) is not a CNF expression but a *pseudo Boolean* expression. Much work has, however, been done to translate between these two types; see [45]. Here, a standard approach is applied. First new Boolean variables y_{rt} are introduced which take on the value true if x_{rtj} is true for any $j \in \mathcal{J}_{rt}$, which is established by the CNF expression

$$\bigwedge_{(r,t)\in S^i} \left(\bigwedge_{j\in\mathcal{J}_{rt}} y_{rt} \vee \neg x_{rtj} \right). \tag{3.32}$$

With these new variables, the constraint (3.31) simply states that one of them must be false, which is establishes by adding the condition

$$\bigwedge_{(r,t)\in S^i} \neg y_{rt} \tag{3.33}$$

to MiniSat.

4

Implementation

In Section 3, an overview of the algorithm was given and the parts of the algorithm were motivated and connected to the mathematical tools introduced in Section 2. This Section focus on details; presenting and motivating the particular choices made in the algorithm. Beginning in Section 4.1 with the software used, Section 4.2 provides particulars on how the sets $\mathcal{D}_{rtj}$ are created. Section 4.3 gives more details on the material presented in Section 3.3, while Section 4.4 explains the implementation of the ideas described in Section 3.4.

4.1 Software

The implementation of the algorithm is quite complex and relies on code developed, the commercial software IPS, and several open source packages. The overall performance of the algorithm is heavily dependent on the performance of each of its components.

4.1.1 C++

The algorithm is mainly written in C++; although major parts are composed by libraries; see Sections 4.1.3–4.1.5. C++ was considered a preferable language since the main software IPS has an interface to C++, but also due to its high performance.

4.1.2 Industrial Path Solutions and Lua

Industrial Path Solutions (IPS) is the software that the entire algorithm is built upon. IPS offers a CAD-like environment, where the scene contains the robot objects and a surrounding environment, including a workpiece with the tasks to be performed. IPS contains functions (as noted in Section 1.3) to balance the load of the robots, sequence the tasks, and create paths for the robots between the tasks. Collisions between robots are handled by synchronisation messages; see [46] for more information on IPS.

The algorithm developed in this thesis uses IPS to gather information about the geometry, to build the approximate GVD, to sequence the tasks and plan the robot paths, and to visualise results.

The programming language Lua (see [47]) offers an interface to IPS, which in turn is called from C++. Using this interface, internal functions of the software are called. The most used is the distance function, which computes the distance between

a robot and a point or object, but currently, the Lua interface lacks the possibility to return from what point on the robot this distance is computed. This is the main reason why the distance transform is not used in this project; recall Section 3.4.2.

4.1.3 MiniSat

MiniSat is an open-source SAT-solver, i.e., solves satisfiability problems formulated by expressions in CNF form; recall Section 2.1. MiniSat is written in C++, it is provided as a library, and it has an easy-to-use interface; see [32]. The solver is based on a tree search method, including several methods for pruning the tree. For more details on how the MiniSat solver works, see [48]. For an alternative SAT-solver, see [49].

MiniSat is used to solve the approximate problem; recall that the constraints in (3.5) are on CNF form. MiniSat is particularly useful in the branching procedure, where it is desirable to find an initial column fast or to determine whether the problem is infeasible under a specific branch constraint.

4.1.4 Coin-OR

The Common Optimization Interface for Operations Research (Coin-OR) is an open-source software for optimization problems; see [50]. In the project two packages are used: CLP and CBC. CLP is a simplex solver for LP problems and CBC is a branch-and-cut solver for ILP problems. There are software that are superior to Coin-OR in terms of performance, e.g., CPLEX or Gurobi [51, 52], but Coin-OR is used since its accessible due to its open-source nature.

4.1.5 Voro++

To compute Voronoi diagrams (not generalised) the C++ library Voro++ is used; see [40]. Again, the motivation to favour this over other software is since it is simple to incorporate in the algorithm. For three or higher dimensions, usually convex hull algorithms are suggested for computing the Voronoi diagram; see [19, p. 278]. E.g., the software Qhull (see [39] and Section 3.4.1) computes Voronoi diagrams using effective convex hull algorithms, and could be used instead of Voro++. There also exist more advanced and developed software (e.g., CGAL [53]). Voro++ is specialised for 3D and searches an increasing neighbourhood of each generating point until all faces of the corresponding cell are found; see [41, Ch. 4.2.2] for details.

4.2 Constraint matrix construction

In order to retrieve the set of pairwise collisions $\mathcal{D}_{rtj}$, defined in (3.1) and used in the optimization problem (3.2), geometry measurements are computed in IPS according to the following. For each robot $r \in \mathcal{R}$, each task $t \in \mathcal{T}$, and each robot position alternative $j \in \mathcal{J}_{rt}$, all other robots at all other tasks are positioned using every alternative. For each such pair of alternatives the shortest distance between the two robots is computed (see Figure 4.1) and checked with respect to the clearance

condition (3.1). The set $\mathcal{D}_{rtj}$ contains all cases for which the clearance condition is violated (since it is symmetric only half of the combinations need to be tested). Whenever two alternatives are measured to have a large enough (determined by the robot design) clearance, then all alternatives corresponding to the same pairs of robots and tasks are assumed meet the clearance limit.

This construction is a large preprocessing step to the algorithm which requires several minutes of computations. An interesting development would be to employ a lazy evaluation of the set $\mathcal{D}_{rtj}$ (see Section 6.1) since this step might test for conflicts that do not arise in the algorithm.

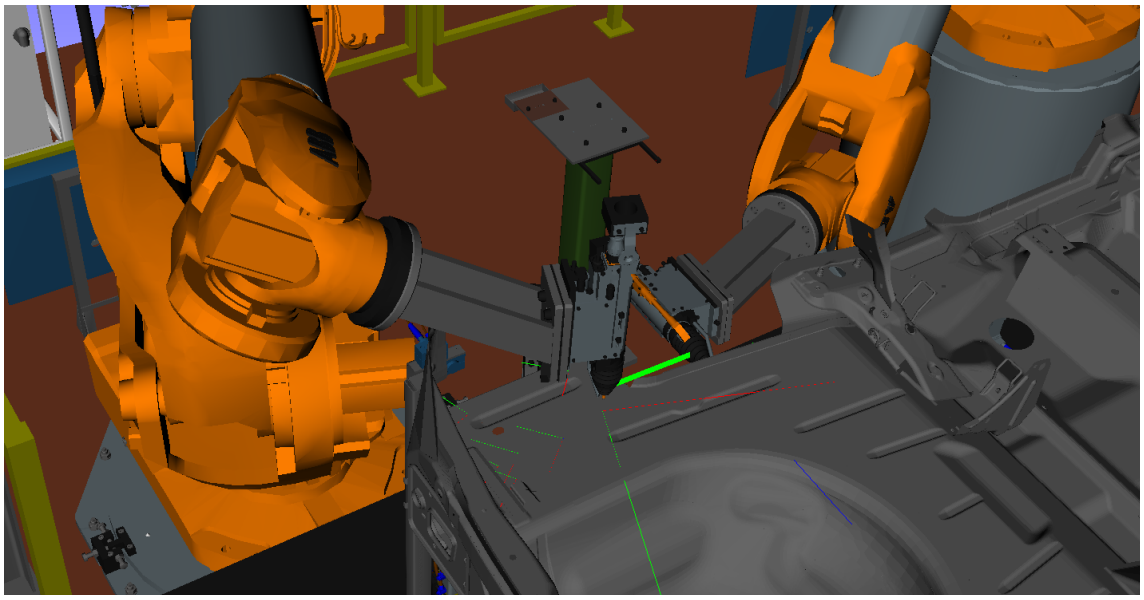


Figure 4.1: Screenshot from IPS illustrating how the collision constraints are constructed. The robots are positioned at adjacent tasks and the green thick line denotes the minimum distance between the robot objects.

4.3 Solving the approximate problem

The approximate problem (3.5), a min-max semi-assignment problem with conflicts constraints, was in Section 3.3 suggested to be solved using a Dantzig-Wolfe decomposition. Three different decomposition approaches are suggested; they are described/detailed in Sections 4.3.2–4.3.4. The branching procedure, which is common to these approaches, is presented in Section 4.3.6. The purpose of solving the approximate problem (3.5) is to find a good approximation of the robot-partitioning surface, described in Section 3.3. The robot-partitioning surface is mainly dependent on the robot-task assignments and secondly on the robot paths between tasks. Thus, the approximate problem only aims to find the assignment and not the sequence of the tasks. The problem (3.5) is a rough estimate of the exact model (3.2) in that the cost function is approximated.

Recall that the constant c_{rtj} in (3.3) denotes the cost of an assignment. We define these costs to be the sum of the task duration t_d and the time t_p used by the

shortest path from the home position to the joint configuration j in the robot joint space. The task duration is the time a robot needs to perform the task: these times are usually longer than the travel times for the robot paths.

The second term t_p has two purposes. First, since all times are measured from the home position, the solution to the approximate problem will favour tasks that are close to the home position and it will in a sense separate the robots. The second reason derives from the first: when the alternatives differ, many of them can be discarded, thus simplifying the search for an optimal solution. Since a branch-and-bound solver may never discard nodes if all solutions are almost equally good, this implies that all solutions need to be tested. Note that the second term is a lower bound on the true travel time from the home position, since neither any static geometry, nor workpiece, nor partitioning surface, is taken into account.

There are of course many alternatives for the cost approximation (3.3), either by a different definition of the assignment costs c_{rtj} or by using a different approximation formula. See, Further work in Section 6.1.

4.3.1 Preprocessing by removing redundant variables

Using the objective function (3.3), many variables can be removed from the approximate problem (3.5). This since, consider two variables x_{rtj_1} and x_{rtj_2}. If it holds that $\mathcal{D}_{rtj_1} \subset \mathcal{D}_{rtj_2}$ and $c_{rtj_1} \leq c_{rtj_2}$, then x_{rtj_2} can be set to 0. This procedure removes a very large number of variables from the problem which speeds up the computations and prevents from unnecessary branching.

4.3.2 Solving MP with simple subproblem

Section 3.3.3 presents a procedure for solving the approximate problem (3.6), where the maximum approximate cycle time (3.3) is treated as a resource to be minimised. In short, the problem (3.5) was first reformulated into the equivalent problem (3.12), and by a linear relaxation, the MP was retrieved. The suggested decomposition resulted in a very simple subproblem, i.e., (3.13). The corresponding solution to MP is retrieved as described in Section 2.3, where RMP is iteratively solved using Coin-OR CLP which provides the optimal dual variable values, which are used in the subproblem.

To find a solution to the IMP (3.12) and hence to the approximate problem, the branching procedure, presented in Section 4.3.6, is used.

4.3.3 Solving MP with min-max subproblem

Motivated by findings presented in [14] it may be superior to solve the approximate problem using the decomposition suggested in Section 3.3.4. The solution procedure is similar to that presented in Section 4.3.2 but instead the IMP (3.16) and the subproblem (3.17) are considered. As suggested in Section 3.3.4, the subproblem is solved using a Lagrangian relaxation and a subgradient algorithm.

The subgradient algorithm proposed here differs a little from ordinary subgradient methods, which in each iteration finds a subgradient and decides on a step

length (see [34, Ch. 6.4]). These are very successfully in solving large-scale problems but since the number of robots is usually rather small, another algorithm is here proposed, sometimes referred to as a bundle method (see [34, p. 174]).

A subgradient $\mathbf{g} \in \mathbb{R}^m$ to a concave function $q\colon \mathbb{R}^m \mapsto \mathbb{R}$ in a point $\boldsymbol{\mu}_0 \in \mathbb{R}^m$, is a vector satisfying the relation

$$q(\boldsymbol{\mu}) \leq q(\boldsymbol{\mu}_0) + \mathbf{g}^{\mathrm{T}}(\boldsymbol{\mu} - \boldsymbol{\mu}_0), \tag{4.1}$$

i.e., it defines a half-space, in which the values of q is lower than its value in the current point. To find a subgradient of the objective function in the maximisation problem (3.20), we use that it is the Lagrangian dual from the relaxation of constraint (3.18b) in problem (3.18). According to [34, p. 180] the subgradient $\mathbf{g}$ at the point $\boldsymbol{\mu}$ for component $r \in \mathcal{R} \setminus \{\tilde{r}\}$ becomes $\mathbf{g}_r := -\mathbf{C}_{\tilde{r}}^{\mathrm{T}}\mathbf{x}_{\tilde{r}}^k + \mathbf{C}_r^{\mathrm{T}}\mathbf{x}_r^k$, where $\mathbf{x}^k$ minimise the Lagrangian function in (3.20).

To solve the maximisation problem (3.20), i.e., the Lagrangian dual problem of the subproblem (3.18) we introduce Algorithm 4.1, where $q(\boldsymbol{\mu})$ is the Lagrangian dual function and $\mathbf{g}(\mathbf{x})$ is defined as above. It is inspired by a divide-and-conquer strategy, where the space containing the solution $\boldsymbol{\mu}^*$ is iteratively reduced by computing subgradients and cutting off half-space. The efficiency of such an algorithm reduces quickly with an increasing dimension of the space (number of robots), since then more half-spaces need to be considered in each iteration. However, experiments showed that it is superior to a standard subgradient algorithm—e.g., divergent series step length rule described in [34, p. 173]—for this low-dimensional problem (less than ten robots).

Algorithm 4.1 Divide-and-conquer by subgradients

1: Initialise $k := |\mathcal{R}| - 1$, $\boldsymbol{\mu} := \mathbf{1} \in \mathbb{R}^k$, $\mathbf{A} := -\mathbf{I}$, and $\mathbf{b} := \mathbf{0} \in \mathbb{R}^k$.
2: Evaluate the Lagrangian dual function $q(\boldsymbol{\mu})$ and let $\mathbf{x}$ denote the corresponding primal solution.
3: Compute new cut $\mathbf{A}_{k+1} := \mathbf{g}(\mathbf{x})^{\mathrm{T}}$ and $\mathbf{b}_{k+1} := \mathbf{A}_{k+1}\boldsymbol{\mu}$.
4: Determine a search direction $\mathbf{s} := \mathbf{g}(\mathbf{x}) + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is a small random perturbation.
5: Find longest feasible step d such that $\bar{\boldsymbol{\mu}} := \boldsymbol{\mu} + d\mathbf{s}$, by checking all constraints given by $\mathbf{A}\bar{\boldsymbol{\mu}} \leq \mathbf{b}$. If d is unbounded then set $d := d_{\mathtt{max}}$ ($d_{\mathtt{max}}$ is a parameter to tune).
6: Update $\boldsymbol{\mu} := \boldsymbol{\mu} + \frac{d}{2}\mathbf{s}$ and $k := k + 1$. If $k < k_{\mathtt{max}}$ return to step 2.
7: Return the $\mathbf{x}$ with best primal objective encountered during the process.

Since each iteration introduce a new condition dividing the considered set. Each introduced half-space will reduce the size of the considered set down to half its original volume; hence, if the space is of dimension d the diameter is reduced by approximately the fraction $\frac{1}{2}^{\frac{1}{d}}$ indicating that the algorithm is suitable for low-dimensional problems (tested $d \leq 10$). Experiments showed that the perturbation $\boldsymbol{\varepsilon}$ introduced in step 4 prevented the algorithm from adding very similar constraints and thus not finding good solution within a reasonable value of k_{max}.

4.3.4 Solving MP with conflicted subproblem

The procedure to solve the approximate problem (3.6) using the conflicted subproblem was introduced in Section 3.3.5. It was noted that many redundant variables may be removed using a technique similar to the preprocessing step described in Section 4.3.1.

In the subproblem (3.23) each task needs to be assigned without conflicts and the goal is to minimise the total sum of the associated variable costs, $-\bar{\gamma}_{rtj}$. Hence, for every pair of variables corresponding to the same task, one of them is redundant whenever the relations $-\bar{\gamma}_{rtj} \leq -\bar{\gamma}_{\bar{r}\bar{t}\bar{j}}$ and $\mathcal{D}_{trj} \subset \mathcal{D}_{\bar{r}\bar{t}\bar{j}}$ hold. Moreover, since each task needs to be assigned by at least one alternative, any element of the set

$$\bigcap_{r\in\mathcal{R}} \bigcap_{j\in\mathcal{J}_{rt}} \mathcal{D}_{rtj} \tag{4.2}$$

may be discarded.

By iteratively enforcing these two conditions on every task until no more variables can be removed from the problem, it is greatly simplified without changing the optimal solution value.

4.3.5 Generating multiple columns

A popular and usually effective method when solving the column generation subproblem is to return more than one column with negative reduced cost in each iteration; see, e.g., [54, 13, 55]. The idea is to reduce the number of times the subproblem needs to be solved. However, as noted in [14], if the subproblem is solved fast, then returning only the best column might be better since it reduces the number of redundant columns in the MP, in turn reducing the complexity. It turns out that the subproblems (3.13) and (3.17) can be solved in a much shorter time than that spent solving the MP; preliminary results show that in either case it is more efficient to generate a single column using either of the subproblems.

However, when solving the MP using the conflicted subproblem almost all time is spent solving the subproblem, since the MP becomes very small. In this case it could be beneficial to generate multiple columns in each iteration; however a simple test showed no signs of a better performance but rather a slight decrease when returning more than a single column. The method tested was to return the k best solutions found in the branch-and-bound search of the subproblem; recall Section 3.3.5.1. Three different problems were tested and no value of k was better than $k = 1$, in terms of time spent in each branching node.

4.3.6 Branch order

In Section 3.3.7, the branching rule was introduced and motivated. To search the resulting tree, a rule for prioritising the branches needs to be introduced. The rule is simply a *depth first search* (see [36]) which always searches the deepest node in the tree. This allows for finding a better integral solution fast and since the tree have a finite depth the part kept in memory will be rather small. Finding good integral solutions are important to allow for a more efficient pruning of the tree.

4.3.7 Improvement heuristics

Some simple improvement heuristics are implemented in different parts of the algorithm.

The first is a local search heuristic aiming to find a better feasible solution to the min-max subproblem (3.17). The neighbourhood is defined as reassigning a single task with any other alternative, mathematically this becomes for a $\mathbf{x} \in \mathbf{S}$ its neighbourhood becomes

$$\mathbf{N}_1(\mathbf{x}) := \left\{ \bar{\mathbf{x}} \in \mathbf{S} \colon \left(\sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{J}_{rt}} \frac{|\bar{x}_{rtj} - x_{rtj}|}{2} \right) \leq 1 \right\}, \tag{4.3}$$

this is here referred to as a 1-neighbourhood (since a task is reassigned). This search is very fast since the 1-neighbourhood is rather small (total number of robot positions m) and evaluating the min-max objective (3.17a) and (3.17b) may be done in $\mathcal{O}(m)$ time.

The second improvement heuristic is also a local search heuristic, which is applied to the initial solution provided by MiniSat when solving the approximate problem (3.5). Again, the 1-neighbourhood defined in (4.3) is used. However, here also the feasibility to the conflict constraint (3.2c) and the old solution constraint (3.31) needs to be checked, i.e.,

$$\tilde{\mathbf{N}}_1(\mathbf{x}) := \{ \bar{\mathbf{x}} \in \mathbf{N}_1(\mathbf{x}) \colon \text{(3.2c) and (3.31) hold} \}. \tag{4.4}$$

The last heuristic is a feasibility heuristic applied to any fractional solution found for the MP (3.12), (3.16) or (3.22). It iteratively searches for the variable possessing the highest fractional value. This variable is set to 1, while all other variables corresponding to the same task as well as all variables in conflict with this variable are set to 0. When all fractional values have been removed, there might be unassigned tasks, for each of which the best feasible alternative is set to 1. This process continues until each task is assigned to a robot or no feasible solution can be found.

4.4 Approximation of GVD

In Section 3.4, several variants were introduced to approximate the GVD of the robot objects, simultaneously positioned at each of the tasks assigned to them by the approximate problem solution (3.5). The first algorithm is based on sampling points on these objects and then uses a standard Voronoi diagram from Voro++ as the approximation.

The second algorithm is based on the work in [28] and uses a distance field to approximate the GVD. Section 4.4.2 illustrates this method and the suggested improvement of it.

4.4.1 Approximation by points

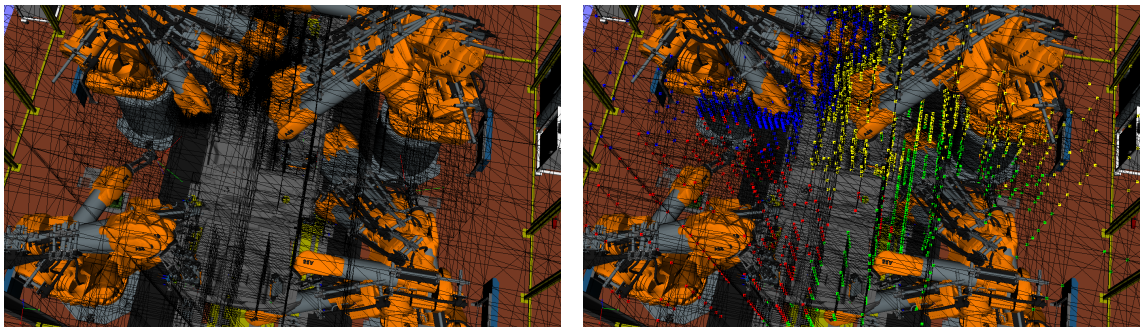
Following Algorithm 3.2, first a number of points P_r are randomly sampled on the robot objects R_r retrieved from the solution to the approximate assignment

problem (3.5). The Voronoi diagram of these points is computed using Voro++; then, every face shared between two Voronoi cells with generator points on different robot objects will be part of the approximation of the GVD.

One thing remains in order to build these faces in IPS: the faces from Voro++ are convex polygons and need to be reduced to a set of triangles. However, since the polygons are convex, this is done by, for a vertex, adding an edge to each vertex with which it is not an immediate neighbour.

4.4.2 Approximation by distance field

Details on approximating the GVD using a distance field are given in Section 3.4.2. The main part of this algorithm is written purely in C++, but all collision checks and distance calls are done in IPS. The first step in the algorithm consists of constructing an octree with the condition that there should be at least one empty cell separating the robot objects; see Figure 4.2a where the empty buffer cells are most visible between the lower two robots. Then distance calls in IPS are used to determine which robot is the closest to each vertex in the octree. Using this information, the octree is further subdivided to resolve any ambiguities in the topology and to ensure that the approximate GVD is contained in empty cells; see Figure 4.2b.

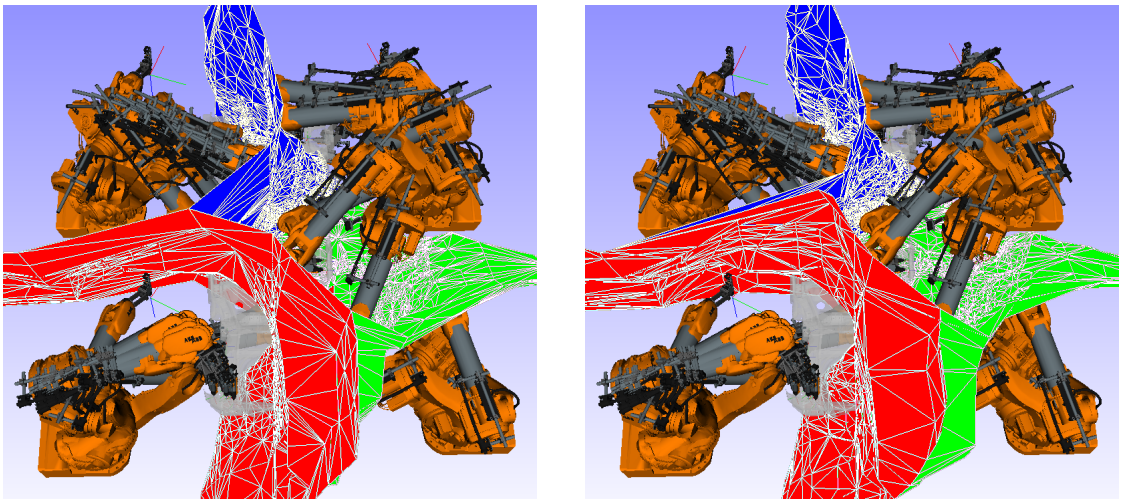


(a) Initial octree. **(b)** Refined octree and closest objects.

Figure 4.2: Screenshots from IPS illustrating the resulting octrees. (a) The initial octree. (b) The octree when all ambiguities have been resolved and the closest robot objects determined.

Sections 3.4.2 and 3.4.3 offer two different interpolation procedures to approximate the GVD, given the distance field. In Figure 4.3a, the result of the first procedure is presented. The motivation for the improvement was to remove certain artefacts; recall Figure 3.7. These artefacts are apparent where the blue and red part of the surface meet. Here the robot to the upper right is assigned too much space. These artefacts results from the face- and cell-midpoints added to simplify the triangulation. But these additional points are not necessarily near the true GVD.

In Figure 4.3b, the result of the suggested improvement is shown. The goal of the improvement is to remove the need of additional points for creating the triangulation, but to only use interpolation in the distance field already acquired. This is done by first using Voro++ to get a Delaunay tetrahedralization of each cell.



(a) The approximate GVD. **(b)** The improved approximation.

Figure 4.3: Screenshot from IPS illustrating the resulting approximation of the GVD. (a) The mesh retrieved using the suggested interpolation formula. (b) The mesh retrieved when a Delaunay tessellation is done in each cell before interpolation.

Then, the approximation of the GVD in each tetrahedron is provided by using the same interpolation formula (3.29) as in the previous case.

5

Tests and results

Here the results of the algorithm are presented, first in detail on the station described in Section 1.5. For this instance, all parts of the algorithm are examined and illustrated. Then, we compare several instances in a quantitative analysis, in which, due to IP restrictions, only some representative numbers of the algorithm performance are presented. In all examples, the clearance limit d_c (3.1) is set to 5cm.

The algorithm was run on a CPU of 2.7GHz on a single core, except for the commercial solver Gurobi that was run in parallel on eight cores.

5.1 Approximate problem

Here, the results comparing the three decomposition methods used to solve the approximate problem (3.5) are compared. The method decompose the MP using the simple subproblem (Section 3.3.3) the min-max subproblem (Section 3.3.4) the conflicted subproblem (Section 3.3.5), respectively. To facilitate the validation of these three methods, the Coin-OR CBC solver (Section 4.1.4) is also used to solve the approximate problem (3.6).

One reason not to decompose the simple subproblem into multiple subproblems is to reduce the number of columns required to solve the master problem (MP). In Table 5.1, the number of columns needed to solve the MP using the different strategies are presented, and compared to the number of original variables available at each node.

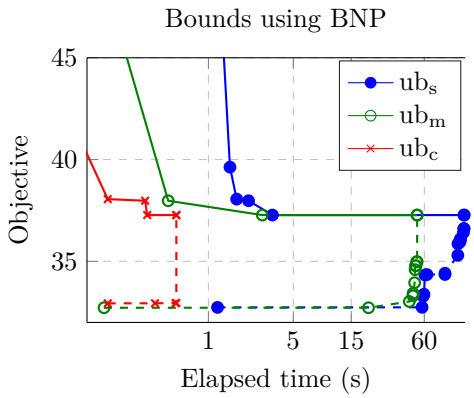
Subproblem	[1st, 2nd, 3rd quartile] of columns	[1st, 2nd, 3rd quartile] of variables
Simple	[11, 16, 34]	[96, 103, 122]
Min-max	[12, 16, 21]	[95, 106, 118]
Conflicted	[7, 9, 13]	[131, 133, 134]

Table 5.1: The left column shows the quartiles of the number of columns needed to solve the MP in each node, using the simple, min-max, and conflicted subproblem, respectively. The right column shows the corresponding quartiles of the number of original variables allowed under the branching rule.

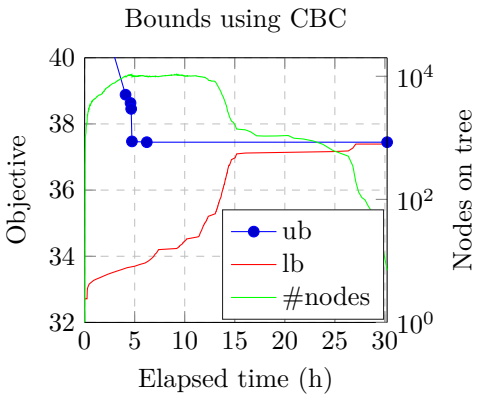
Table 5.1 not only shows that fewer variables need to be regarded but it also reveals that, when using the min-max subproblem, there is a reduced risk of generating too many columns to solve the MP. This fact can be seen since the upper

quartile of the number of columns is not very far from the median. However, the largest difference appears when using the conflicted subproblem: the nodes considered more variables then by using one of the other decompositions, which indicates that the tree is pruned more early on. The conflicted subproblem also needs to generate fewer columns in order to solve the MP; this is expected since the subproblem is harder to solve.

To compare the efficiency of these solution methods, the achieved bounds on the optimal value are plotted versus the time spent in the algorithm; see Figure 5.1a. In the figure we note that a harder subproblem results in better performance, both in terms of finding good integral solutions fast and searching the entire tree. In all cases, the branching order was depth first search, which manifests in the behaviour of the lower bounds: it is updated whenever the tree contains a single leaf.



(a) Upper (solid) and lower (dashed) bounds on the IMP using branch-and-price with the three methods; for all three the search ended when optimality was verified.



(b) Bounds and the size of the branching tree during the solution of the approximate problem (3.6) with Coin-OR ILP solver.

Figure 5.1: Illustration of the bounds achieved when using the different approaches for solving the approximate problem (3.6). Legend entries: simple subproblem $_s$, min-max subproblem $_m$, and conflicted subproblem $_c$.

From Figure 5.1, it seems that the initial lower bounds from the min-max and the simple subproblems are identical, but experiments showed that they are only similar. The lower bound from the conflicted subproblem is however slightly better than both simple and min-max subproblems. Not shown in the figure is that the branching procedure visited roughly 1000 nodes using either the simple- or the min-max subproblem. When using the simple subproblem almost all computation time is spent solving the master problem, resulting in $0.1s$ per node in the branching procedure. This compared to using the min-max subproblem, for which the average computation time in each node was 0.05s, and about half of that time was spent in the subproblem.

The approach using the conflicted subproblem only visited thirteen nodes and spent $0.05s$ on average in each node. The fast computation time is mostly due to the few number of columns generated. The few nodes visited is a result of the stronger bound retrieved by this decomposition.

Figure 5.1b gives a point of reference where Coin-OR CBC solver have been used to solve the approximate problem until optimality. Coin-OR CBC verified the optimal value of 37.45, which agrees with that of the solutions found by all three decomposition methods, as shown in Figure 5.1a. As an additional comparison, the commercial solver Gurobi found an optimal solution in three seconds and verified it optimal in another five seconds. Hence, the approach of using the conflicted subproblem decomposition seems to compete with cutting edge optimisation software; this by utilising problem specific properties.

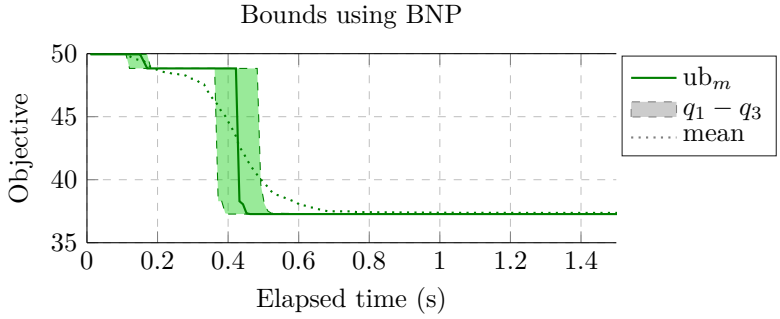
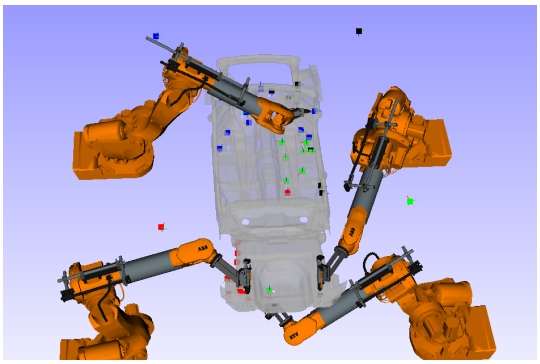
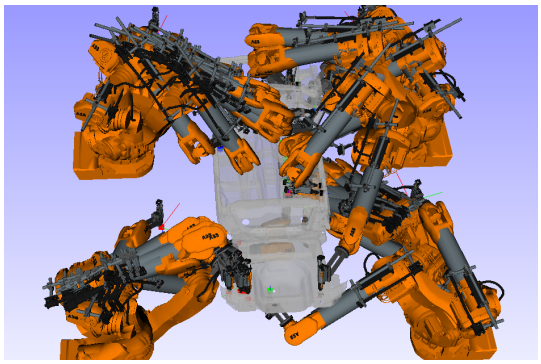


Figure 5.2: Illustration of the stability in solving the IMP using the min-max subproblem. The quartiles (dashed lines), median (solid line), and mean (dotted line) are computed from 200 runs.

In figure 5.1a results for solving the min-max subproblem is made from a single run, however, since Algorithm 4.1 for solving the subproblem is non-deterministic, also the result of the branch-and-price algorithm will be non-deterministic. Figure 5.2 illustrates the spread of the bounds gained by repeatedly solving the subproblem; the small spread indicate that the performance is stable.



(a) Intersection free assignment from the approximate problem.



(b) The robots' unions used in GVD approximation.

Figure 5.3: Illustration of an optimal solution to the approximate problem (3.6) with a clearance limit d_c of 5cm between robots. (a) The robots positioned at one of their assigned task; which are coloured by the assignment. (b) The union of all assignments; the minimal distance between robots' unions is 7.16cm.

Figure 5.3 illustrates the geometry of one of the optimal solutions retrieved. Even though the clearance limit was 5cm this solution has 7.16cm as a shortest distance between the robots' unions. The geometry from Figure 5.3b is used to determine the approximate GVD.

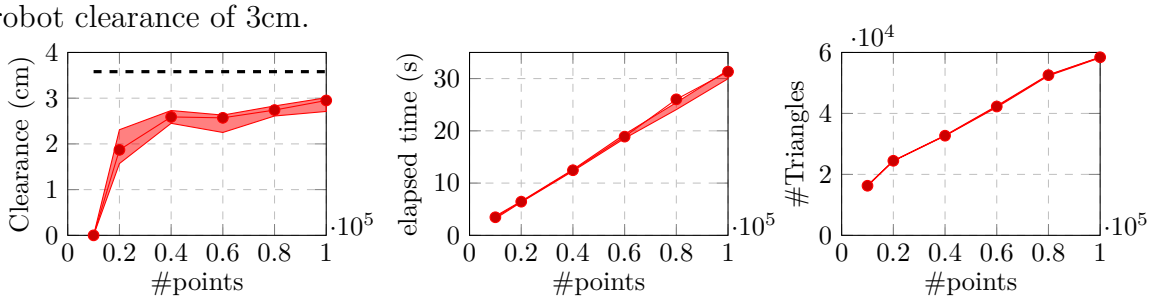
5.2 Generalised Voronoi diagram

Here, the three algorithms for approximating the GVD will be compared with respect to four measures determining the quality of the mesh:

- The minimum clearance between the resulting surface and the robots' unions (see Figure 5.3b). Note that this should equal half of the robot-robot clearance.
- The number of triangles used to approximate the GVD, to measure the adaptivity of the algorithm, i.e., not using too many triangles.
- The CPU time needed to compute the surfaces.
- A visual examination to determine whether the approximations resemble the true GVD or not.

Starting by examining the performance of Voro++, which has a degree of freedom of how well the GVD should be approximated; that is the number of points sampled on the robot unions' surfaces. With sufficient amount of points the approximation would be exact.

These results are presented in Figure 5.4. Note, however, that these results are dependent on the minimum robot-robot clearance in the solution. If the robot-robot clearance is small a larger number of points are required, e.g., in a solution with a minimum robot-robot clearance of 10cm, 10000 points was sufficient to give a minimum surface-robot clearance of 4cm, but in the Figure 5.4a the minimum robot-robot clerance where only 7cm and then 100000 was need to get a surface-robot clearance of 3cm.



(a) Robot-surface clearance.

(b) Time needed to compute the surface.

(c) Complexity of the surface.

Figure 5.4: Illustration of the performance Voro++ used to approximate the GVD, where # points being the number of points randomly sampled on each robot union. In each of the figures, the middle line indicates the median outcome and the coloured area indicates the first and the third quartile of the outcome, as estimated over ten runs. (a) The dashed line denotes half the robot-robot clearance. The figures illustrated in the diagrams correspond to the solution illustrated in Figure 5.3; the resulting GVD using 60000 points is illustrated in Figure 5.5.

Recall Figure 4.3, which illustrates the result of applying the approximation by the distance field approach and the suggested improvement of it. The corresponding Figure 5.5 shows the result (using the same robot geometry) instead using Voro++

and 60000 points on each robot. Note that the artefacts observed in the triangulations using the distance field approach (where some robots where assigned to much space) is not present in the triangulation from Voro++.

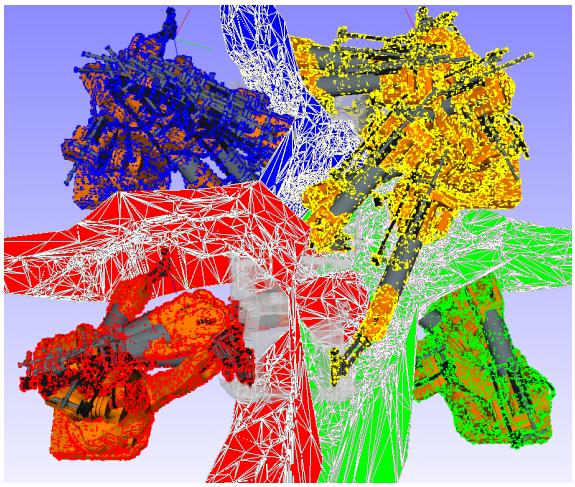


Figure 5.5: The resulting surface from creating the Voronoi diagram of 60000 points randomly sampled on each robot union, illustrated in Figure 5.3b, then only using the part of resulting Voronoi diagram that separates the unions. For details, see Algorithm 3.2. Voro++ is used to compute the Voronoi cells.

Table 5.2 presents the time consumption of the GVD approximation using a distance field and its improvement. Recall that the improved version spent extra effort only to create a good mesh in which to interpolate. In the steps *measure* and *resolve* the major time is spent by IPS for computing point-object distances and in the *measure* step the closest robot object is determined for about 10000 octree vertices, using Algorithm 3.4.

Component	Build octree	Measure	Resolve	Mesh	Improved mesh
Elapsed time [s]	8.98	5.41	12.16	0.98	2.69s

Table 5.2: The time spent approximating the GVD using the distance field approach (Section 4.4.2). The column *improved mesh* replaces the column *mesh* in the suggested improvement. The total time is presented in Table 5.3.

In Table 5.3, the three approaches are compared in terms of clearance, number of triangles, and time to compute. In terms of speed, Voro++ is superior, but it is inferior in terms of precision and number of triangles used. Recall the choice made to not implement a distance transform, Section 3.4.2.2, using this transform the results from the distance field approach and from Voro++ would be similar, see Section 6.1. Moreover, the suggested improvement, which aimed to remove the artefacts present in the approximation by distance field approach, is slightly worse than the original in all aspects of Table 5.3.

	Minimum clearance [mm]	# triangles	CPU time [s]
Voro++ (60000 points/robot)	25.7	42268	19
Approx. using distance field	32.8	15862	28
Improvement of distance field approx.	29.0	18410	30

Table 5.3: Performance of the three approaches applied to the assignment in Figure 5.3b. The robot-robot clearance in this case 71.6mm and hence without any approximation the GVD should have a clearance of 35.8mm.

5.3 Algorithm performance

The complete algorithm described in Section 3.2 is here evaluated. First, results for the example station introduced in Section 1.5 are presented; then we present some quantitative results from the algorithm tested on some other stations.

5.3.1 Tests on the example station

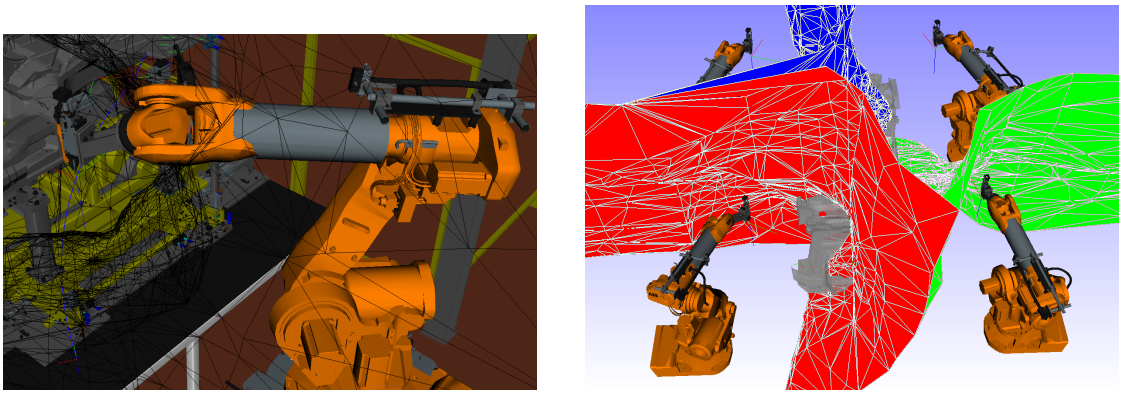
There are four major components in the algorithm: construction of the conflict constraints; solution of the approximate problem; approximation of the GVD; sequencing and path planning by IPS. The time spent in each algorithm component, for the example case, is presented in Table 5.4.

Component	Constraint matrix	Approximate problem	GVD	IPS
Elapsed time [s]	248	0.5	28	~300

Table 5.4: The time spent in each algorithm component during the first iteration (recall Figure 3.1), for the example station illustrated in Figure 5.3a. The approximate problem is solved by decomposing the IMP with the conflicted subproblem. The GVD is approximated using the distance field without the suggested improvement.

For the example station described in Section 1.5, the approximate GVD rarely makes the complete problem infeasible. Figure 5.6a shows a feasible path to be added to the approximate problem and Figure 5.6b show the resulting, feasible GVD approximation. The feasibility of the path is ensured by adding conflict constraints associated with each task (see Section 3.5.1). For this particular case, the new solution contains the assignment corresponding to the ensured path.

The performance of the second feedback procedure (see Section 3.5.2), which restricts the old solution and iteratively resolves the approximate problem to find new and possibly better solutions, is presented in Figure 5.7. It is apparent that the approximation of the complete problem is rather good, since there seems to be a correlation between the approximated and computed cycle time series, but rather many solutions are required before the approximate cycle time becomes unreasonably high. Since only the GVD—and no other partitioning surface—is considered, we cannot verify that the solution found is optimal to the complete problem (see



(a) The feasible path to be added.

(b) The new feasible GVD approximation.

Figure 5.6: Conflict constraints added to the approximate problem, where the entire path must satisfy the clearance limit. (a) A snapshot of the robot on the added path to the inaccessible task. Note that the robot intersects the current GVD approximation, illustrated by black lines. (b) The solution and the approximate GVD to this new problem. Note that the lower left robot, magnified in (a), receive more space due to the added path constraint.

Section 3.5.2). Nevertheless, assuming that the GVD equals the optimal partitioning surface, the solution found seems to be near optimal in the approximate problem.

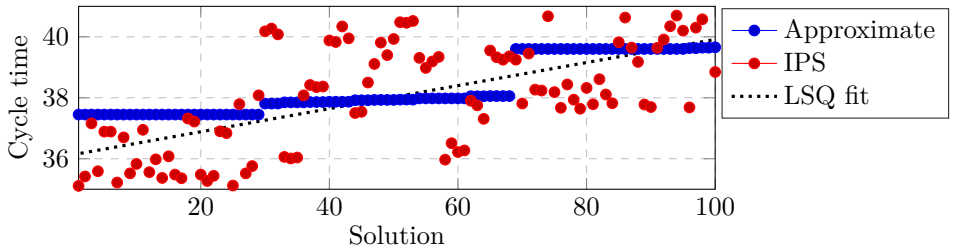


Figure 5.7: The approximate cycle time and the cycle time as computed by IPS for every solution in the iterative process (Figure 3.1). The dotted line is a linear least square fit to the IPS cycle times. The algorithm applied to the example station illustrated in Figure 1.1.

Figure 5.7 illustrate how the solutions to the approximate problem are found in the order of increasing objective value. An additional check that indeed all solutions were found, the same procedure was done in Gurobi, producing the same solutions. Hence, the algorithm passes this stability test.

We conclude the results for this example station: For our approach, using the partitioning constraints, the shortest cycle time found was 35.11 seconds. For the current approach, in which IPS uses a synchronisation of the robot movements to prevent collisions, the shortest cycle time found was 33.53 seconds (which should be less, since the partitioning constraints are relaxed). Hence, applying our algorithm yields partitioned robust robot movements at the cost of an increased cycle time by slightly less than 5%.

5.3.2 Tests on an additional set of stations

Finally, the algorithm is tested on an additional set of stations. Table 5.5, gives some statistics of these stations and on the performance of the algorithm on these stations. Station 1 is the reoccurring example, while stations 2 and 3 are new. Here only the number of robots, task, and alternatives of the stations can be stated, this since they are in use in industry and there is an economical motivation for a non-transparency.

Problem data/statistics	Station 1	Station 2	Station 3
#Robots	4	4	10
#Tasks	48	43	201
#Alternatives	1092	1825	4882
Time in approx. problem/iter [s]	0.5	57	172800
Time in GVD approx./iter [s]	28	43	633
#iterations	100	100	1
Cycle time increase [%]	4.6	2.6	-1.0
slope [s/iter]	0.038	-0.002	-

Table 5.5: Data and result statistics for the three stations to which the algorithm was applied. Cycle time increase refers to the cycle time achieved with the partition constraint compared to using IPS, without the partitioning constraint. The solution of the approximate problem for station 3 was stopped after 172 800 seconds (48h), without an optimal solution found. Slope denotes the derivative of the line fitted to the cycle times vs. solution number, cf. Figure 5.7. Since only one iteration was done for Station 3 no slope could be computed.

From Table 5.5 we see that the algorithm is rather successful in finding partitioned solutions with small increases in cycle time, compared to the corresponding synchronised solutions. A notable failure is that the algorithm failed to search the entire tree on Station 3. The reason for this failure is that station 3 consists of three separate robot cells constructing a single workpiece. This means that there are too many solutions whose approximate solution value differ too little and hence it is not possible to prune the tree at an early enough stage. Thus when working with multiple separate robot cells the approximate cost is too similar between different solutions, despite the attempt made in Section 4.3. However, by using Gurobi we were able to verify that the solution found by the branching procedure is optimal.

Station 2 contained fewer conflicts (due to a different robotic tool) and hence more branches needed to be visited in order to search the tree; this is the cause of the long solution time for the approximate problem. The different tool —allowing for more non-conflicting alternatives— is also the cause of the longer time to compute the GVD, which contained more cells and both the cell intersection and distance computation were somewhat slower.

A last thing to note for station 2 is the slightly negative slope, which indicates that better solutions may still be found and that many more iterations are required before a solution can be considered close to optimal. This is also understood from the fact that the best solution was found on iteration 96. This since, in station 2, the

duration of performing a task is much less and thus travel time becomes dominant. In this case, the current approximation of the objective function is not enough good, resulting in the negative slope.

The cycle time decreases in station 3 which, to the best of our reasoning, has three possible explanations:

- Station 3 is composed by three separate robot cells with the workpiece moving between them, allowing some tasks on the workpiece to be reached by several robots. This yields an extra degree of freedom in the task sequencing problem that decreases the impact of bad partitions.
- In a synchronised robot program, a robot waits until the planned path is clear. A faster but longer collision-free path (with zero waiting time) may, however, exist.
- Partitioning a robot station into one distinct volume per robot results in a computationally easier path optimization problem (to be solved by IPS). Hence, for very large problem instances, the partition facilitates the search for good, feasible solutions.

6

Conclusion

We have considered the problem of minimizing cycle time while avoiding collisions within a robot station consisting of multiple industrial robots that collectively should cover a set of tasks on a workpiece. In order to prevent the robots from colliding we have investigated an alternative to synchronise the robot programs by adding synchronisation/interlocking signals. The alternative approach was to partition the space within the station, separating the robots. To achieve this, we solved an approximate problem using a Dantzig-Wolfe decomposition, yielding an assignment of the tasks to the robots. Using this assignment, a partition was made by the GVD concept and using this also the sequence and the paths between the tasks could be planned using IPS. This process was then iterated, in each new iteration new constraints were added to the approximate problem to prevent infeasible and previous solutions.

From the results, we conclude that this algorithm is successful in partitioning the space while keeping the cycle time within acceptable limits. In addition, since the conflict constraint only allow a few good solutions, the idea to repeatedly solve an approximate problem in order to find candidate solutions to the complete problem seems successful. For very large problem instances consisting of several production cells, implying that many good candidate solutions exist, our algorithm still performs well. This is due to the fact that tasks may then be swapped between cells, resulting in less impact from bad partitions.

We also showed how the min-max semi-assignment problem with conflict constraints can be solved using a branch-and-price procedure; the best decomposition of the optimization problem was to keep the load balancing in the MP and include the conflict constraints in the subproblem. This procedure was shown to be more efficient than using a standard branch-and-cut solver, and slightly more efficient than a cutting edge branch-and-cut solver. For very large problem instances, the solution found to the approximate problem was verified optimal by Gurobi, but not by our algorithm. Hence, there is a potential of development of our algorithm for the approximate problem.

We observed that the generalised Voronoi diagram worked well in a partitioning algorithm and that the Voronoi cells can be computed within a reasonable amount of time. Even though the computation time for the ordinary Voronoi diagram is superior to using the GVD, we prefer the distance field for approximating the GVD, since it is better adapted for closely spaced objects and since the use of the distance transform may also speed up the computations.

6.1 Future work

The algorithm developed in this thesis does not guarantee that a feasible solution is found even if the complete problem (3.2) has a solution, due to the fact that a particular path is added by the feedback procedure. A remedy of this weakness is to add a path for one robot that interferes as little as possible with the paths of the other robots, or to add a path that is in the solution to the complete problem.

The iterative process prevents all similar solutions —identical assignment of the tasks to the robots but disregarding the specific alternative— to occur in the approximate problem, this with the argument that IPS will consider all alternatives feasible by the computed GVD. However, the approximate GVD is computed from the specified alternatives and hence the solution from IPS will depend on the particular assignment found in the approximate problem. This should somehow be prevented, either by brute force; only preventing the particular solution found in each iteration from reoccurring, or by not using the GVD concept and instead finding an optimal partitioning surface for a robot-task assignment without any alternative specified.

The GVD is here assumed to be the optimal partitioning surface for a solution to the approximate problem. This might not be true and the cycle time might reduce if a better surface could be generated. An idea is to iteratively improve the surface, e.g., by finding weights in a weighted Voronoi diagram that minimise the cycle time or by iteratively recompute the GVD for the complete cycle, using the swept robot volumes as objects to partition.

One suggested improvement is to use a better approximation of the complete problem, e.g., by defining c_{rtj}, as the true travel times. Another interesting aspect is to use some other measure than the distance between the home position and the tasks. This could maybe be done by using another model, e.g. a minimum spanning tree, where the edges represents paths between tasks.

If an alternative approximation of the complete problem yields a tight lower bound on its optimal value, then the partitioning and evaluation of candidate solutions should continue until the lower bound exceeds the best upper bound for the complete problem. This might be achieved by using another model of the complete problem (3.2), e.g., some vehicle routing model [56].

An additional gain of using a less approximate model of the complete problem would be that fewer candidate solution would be sufficient, since a better model would preferable provide better solutions. This is most interesting for problem instances with short task duration time, where the current approximation performs poorly.

A minor issue in the current algorithm, is the substantial preprocessing step of computing the conflict constraint matrix. In this step all alternatives to perform a task is compared with all other alternatives to perform any other task with another robot, to check whether the robot-robot clearance limit is satisfied. Probably not all of these alternatives are needed, and it would be interesting to introduce some lazy evaluation, to test the limits when needed. This, however, requires some revision of the solution procedure for the approximate problem in Section 3.3 since there all conflicts constraints are assumed to be known when specifying the subproblem.

Our approximation of the GVD using the distance field diverts from the algo-

rithm suggested by [28], which uses the distance transform to determine the distance to the robot unions. This is mainly due to a current lack in the IPS interface and it is an interesting development of our algorithm, since it will approximate the GVD in much less time than the current version.

As a last remark, in this entire project the robot is assumed to be stationary when performing the task. For other types of tasks, which requires a sweeping motion during the task operation, an idea is to incorporate these tasks in the existing algorithm by letting the volume swept by the robots during the task operation define the object that should satisfy the clearance limit in the conflict constraint of the task alternative.

Bibliography

[1] Fraunhofer-Chalmers Research Centre for Industrial Mathematics; 2016
Available from: `http://www.fcc.chalmers.se/about/` [accessed 2016-01-22].

[2] Almström P, Andersson C, Mohammed A, Winroth M. Achieving sustainable production through increased utilization of production resources. In: 4th Swedish Prod. Symp.; 2011. pp. 398–406.

[3] Segeborn J, Segerdahl D, Ekstedt F, Carlson JS, Carlsson A, Söderberg R. A generalized method for weld load balancing in multi-station sheet metal assembly lines. In: ASME 2011 International Mechanical Engineering Congress and Exposition. American Society of Mechanical Engineers; 2011. pp. 491–499.

[4] Parsaeian S. Implementation of a framework for restart after unforeseen errors in manufacturing systems. Master Thesis, Chalmers University of Technology, Department of Signals and Systems; 2014.

[5] Segeborn J, Segerdahl D, Carlson JS, Carlsson A, Söderberg R. Load balancing of welds in multi-station sheet metal assembly lines. In: ASME 2010 International Mechanical Engineering Congress and Exposition. American Society of Mechanical Engineers; 2010. pp. 625–630.

[6] Spensieri D, Carlson JS, Ekstedt F, Bohlin R. An iterative approach for collision free routing and scheduling in multirobot stations. IEEE Transactions on Automation Science and Engineering. 2015; PP(99):1–13.

[7] Gutin G, Punnen AP. The Traveling Salesman Problem and its Variations. vol. 12. New York: Springer; 2007.

[8] Besanko D, Dranove D, Shanley M, Schaefer S. Economics of Strategy. John Wiley & Sons; 2009.

[9] Pinch R. Conjunctive normal form. In: Encyclopedia of Mathematics. The European Mathematical Society; 2014
Available from: `http://www.encyclopediaofmath.org/index.php?title=Conjunctive_normal_form&oldid=35078` [accessed 2016-02-16].

[10] Nicholson J. De Morgan's laws. In: The Concise Oxford Dictionary of Mathematics. 5th ed. Oxford University Press; 2014
Available from: `http://www.oxfordreference.com/view/10.1093/acref`

/9780199679591.001.0001/acref-9780199679591-e-778 [accessed 2016-02-17].

[11] Lasdon LS. Optimization Theory for Large Systems. Mineola, N.Y: Dover Publications; 2002.

[12] Lundgren J, Värbrand P, Rönnqvist M. Optimeringslära. 3rd ed. Lund: Studentlitteratur; 2008.

[13] Lübbecke ME, Desrosiers J. Selected topics in column generation. Operations Research. 2005; 53(6):1007–1023.

[14] Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH. Branch-and-price: column generation for solving huge integer programs. Operations Research. 1996; 46:316–329.

[15] Vanderbeck F, Wolsey LA. An exact algorithm for IP column generation. Operations Research Letters. 1996; 19(4):151–159.

[16] Geoffrion AM. Lagrangean relaxation for integer programming. In: Balinski ML, editor. Approaches to Integer Programming. Berlin, Heidelberg: Springer Berlin Heidelberg; 1974
Available from: http://dx.doi.org/10.1007/BFb0120690.

[17] Vanderbeck F. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research. 2000; 48(1):111–128.

[18] Desrosiers J, Lübbecke ME. Branch-price-and-cut algorithms. In: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc.; 2010
Available from: http://dx.doi.org/10.1002/9780470400531.eorms0118.

[19] Okabe A. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. 2nd ed. Wiley; 2000.

[20] Descartes R. Le monde de Mr. Descartes ou le traité de la lumière, et des autres principaux objets des Sens: avec un Discours du Mouvement Local, et un autre des Fièvres, composez selon les principes du mème auteur. Chez T. Girad; 1644.

[21] Watson DF. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. The Computer Journal. 1981; 24(2):167–172
Available from: http://comjnl.oxfordjournals.org/content/24/2/167.abstract.

[22] Aurenhammer F. Voronoi Diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys. 1991 Sep; 23(3):345–405
Available from: http://doi.acm.org/10.1145/116873.116880.

[23] Musin OR. Properties of the Delaunay triangulation. In: Proceedings of the Thirteenth Annual Symposium on Computational Geometry. SCG '97. New York, NY, USA: ACM; 1997. pp. 424–426
Available from: http://doi.acm.org/10.1145/262839.263061.

[24] Chew LP. Guaranteed-quality Delaunay meshing in 3D (short version). In: Proceedings of the Thirteenth Annual Symposium on Computational Geometry. SCG '97. New York, NY, USA: ACM; 1997. pp. 391–393
Available from: http://doi.acm.org/10.1145/262839.263018.

[25] Mehlhorn K, Meiser S, O'Dunlaing C. On the construction of abstract Voronoi diagrams. Discrete & Computational Geometry. 1991; 6(2):211–224.

[26] Boissonnat JD, Wormser C, Yvinec M. Curved Voronoi diagrams. In: Boissonnat JD, Teillaud M, editors. Effective Computational Geometry for Curves and Surfaces. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. pp. 67–116
Available from: http://dx.doi.org/10.1007/978-3-540-33259-6_2.

[27] Hoff KE III, Culver T, Keyser J, Lin M, Manocha D. Fast computation of generalized Voronoi diagrams using graphics hardware. In: Proceedings of the Sixteenth Annual Symposium on Computational Geometry. SCG '00. New York, NY, USA: ACM; 2000. pp. 375–376
Available from: http://doi.acm.org/10.1145/336154.336226.

[28] Edwards J, Daniel E, Pascucci V, Bajaj C. Approximating the generalized Voronoi diagram of closely spaced objects. Computer Graphics Forum. 2015; 34(2):299–309.

[29] Burkard RE, Dell'Amico M, Martello S. Assignment Problems. Revisition ed. Philadelphia: Society for Industrial and Applied Mathematics; 2012.

[30] Martello S, Toth P. Generalized assignment problems. In: Ibaraki T, Inagaki Y, Iwama K, Nishizeki T, Yamashita M, editors. Algorithms and Computation: Third International Symposium, ISAAC'92 Nagoya, Japan, December 16–18, 1992 Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg; 1992. pp. 351–369
Available from: http://dx.doi.org/10.1007/3-540-56279-6_88.

[31] Cattrysse DG, Wassenhove LNV. A survey of algorithms for the generalized assignment problem. European Journal of Operational Research. 1992; 60(3):260–272
Available from: http://www.sciencedirect.com/science/article/pii/037722179290077M.

[32] Eén N, Sörensson N. MiniSat; 2016
Available from: http://minisat.se [accessed 2016-02-17].

[33] Larsson T, Patriksson M. Global optimality conditions for discrete and nonconvex optimization: with applications to Lagrangian heuristics and column

generation. Operations Research. 2006; 54(3):436–453
Available from: http://www.jstor.org/stable/25146982.

[34] Andréasson N, Evgrafov A, Patriksson M, Gustavsson E, Önnheim M. An Introduction to Continuous oOtimization. 2nd ed. Lund: Studentlitteratur; 2013.

[35] Ryan DM, Foster BA. An integer programming approach to scheduling. Computer scheduling of public transport. 1981; pp. 269–280.

[36] Clausen J. Branch and bound algorithms—principles and examples. Department of Computer Science, University of Copenhagen. 1999; pp. 1–30.

[37] Hoff KE, III, Keyser J, Lin M, Manocha D, Culver T. Fast computation of generalized Voronoi diagrams using graphics hardware. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co.; 1999. pp. 277–286.

[38] Teichmann M, Teller S. Polygonal approximation of Voronoi diagrams of a set of triangles in three dimensions. Tech Rep 766, Lab of Comp. Sci., MIT; 1997.

[39] Barber CB, Dobkin DP, Huhdanpaa H. The Quickhull algorithm for convex hulls. ACM Trans on Mathematical Software. 1996; 22(4):469–483
Available from: http://www.qhull.org.

[40] Rycroft CH. Voro++: A three-dimensional Voronoi cell library in C++. Chaos. 2009 Dec; 19(4):041111
Available from: http://scitation.aip.org/content/aip/journal/chaos/19/4/10.1063/1.3215722.

[41] Rycroft CH. Multiscale Modeling in Granular Flow. PhD thesis, Dept. of Math., MIT; 2007
Available from: http://hdl.handle.net/1721.1/41557.

[42] Danielsson PE. Euclidean distance mapping. Computer Graphics and Image Processing. 1980; 14(3):227–248
Available from: http://www.sciencedirect.com/science/article/pii/0146664X80900544.

[43] Frisken SF, Perry RN. Simple and efficient traversal methods for quadtrees and octrees. Journal of Graphics Tools. 2002; 7(3):1–11.

[44] Natarajan BK. On generating topologically consistent isosurfaces from uniform samples. The Visual Computer. 1994; 11(1):52–62
Available from: http://dx.doi.org/10.1007/BF01900699.

[45] Eén N, Sorensson N. Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation. 2006; 2:1–26.

[46] Industrial Path Solutions Sweden AB. IPS; 2016
Available from: http://industrialpathsolutions.se/ [accessed 2016-01-22].

[47] Roberto I, Waldemar C, Luiz H. The Programming Language Lua; 2016
Available from: http://www.lua.org/ [accessed 2016-01-25].

[48] Sorensson N, Een N. Minisat v1. 13-a SAT solver with conflict-clause minimization. SAT 2005 Competition. 2005; 2005:53.

[49] Silva JaPM, Sakallah KA. GRASP—a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design. ICCAD '96. IEEE Computer Society; 1997. pp. 220–227
Available from: http://dl.acm.org/citation.cfm?id=244522.244560.

[50] Lougee-Heimer R. The common optimization interface for operations research: Promoting open-source software in the operations research community. IBM Journal of Research and Development. 2003; 47(1):57–66.

[51] IBM. IBM ILOG CPLEX Optimization Studio; 2016
Available from: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/ [accessed 2016-08-04].

[52] Gurobi Optimization, Inc . Gurobi Optimizer Reference Manual; 2015
Available from: http://www.gurobi.com.

[53] The CGAL Project. CGAL User and Reference Manual. 4.7 ed. CGAL Editorial Board; 2015
Available from: http://doc.cgal.org/4.7/Manual/packages.html.

[54] Desrosiers J, Soumis F, Desrochers M. Routing with time windows by column generation. Networks. 1984; 14(4):545–565
Available from: http://dx.doi.org/10.1002/net.3230140406.

[55] Feillet D. A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR. 2010; 8(4):407–424
Available from: http://dx.doi.org/10.1007/s10288-010-0130-z.

[56] Toth P, Vigo D. Vehicle routing problem. Philadelphia: SIAM; 2002.