Robotic task scheduling for industrial applications

Antoine Reot

▶ To cite this version:

Antoine Reot. Robotic task scheduling for industrial applications. Robotics [cs.RO]. 2021. hal-03348001

HAL Id: hal-03348001
https://laas.hal.science/hal-03348001

Submitted on 17 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Ingénieur de l'Ecole Nationale de l'Aviation Civile

Antoine Réot

IENAC18 AVI

End of Studies Project Memoir

March 2021 - August 2021

Robotic task scheduling for industrial applications

2021

Acknowledgements

I would like to thank all the people who helped me successfully complete this internship.

Firstly, I would like to thank my internship supervisor, **Florent Lamiraux**, for his welcome, as well as for all the help and advice he provided all along the internship. His guidance allowed me to carry out this internship by always being there to answer my questions and indicate me the best directions to take.

I would like to acknowledge the help and feedback delivered by **Cyril Briand** and **Christian Artigues**, who were always there to discuss my results and who allowed me sufficient autonomy to explore different paths on my wills.

I would also like to thank my ENAC tutor, **Nicolas Barnier**, who helped me manage my internship and gave me advice related to its pedagogic as well as professional aspects.

Finally, I would like to thank my friends and family for their strong encouragement and support all along the internship.

Résumé

Dans le contexte d'une part grandissante de l'automatisation dans l'industrie, ce stage étudie différentes approches de résolution de problèmes de séquencement de tâches, où un robot mobile est utilisé pour ébavurer les trous d'une pièce à usiner. Sachant qu'un nombre infini de configurations du robot peut en théorie atteindre un trou donné, nous modélisons ce problème sous la forme d'un Problème de Voyageur de Commerce Généralisé (GTSP). Nous explorons alors différentes façons de résoudre ce GTSP, que ce soit en partant d'un large ensemble de positions candidates pour la base du robot, ou bien depuis un large ensemble de configurations de bras du robot avec un choix préalable des positions de la base du robot. Nous implémentons par ailleurs plusieurs variantes à ces deux différentes approches en vue de comparer toutes les solutions et d'évaluer leur avantages et défauts dans un cadre industriel.

Mots clés : Robotique ; Séquencement de tâches ; Optimisation ; Problème de Voyageur de Commerce Généralisé

Abstract

As automation is being used increasingly in the industry, this internship explores ways of solving task scheduling problems using a mobile robot in charge of deburring the holes of a part to machine. As an infinite amount of robot configurations can theoretically reach a given hole, we model this problem as a Generalised Travelling Salesman Problem (GTSP). We therefore explore several ways of solving this GTSP, whether it is from a large initial set of base poses or from a large set of arm configurations where the base poses are chosen beforehand. We also implement several variants of these two different approaches and compare all the described methods in order to evaluate their strengths and weaknesses from an industrial point of view.

Keywords: Robotics ; Task scheduling ; Optimisation ; Generalised Travelling Salesman Problem

---

Contents

Acknowledgements 3

Résumé 4

Abstract 5

1 Introduction **11**
1.1 Subject and context 12
1.2 Brief description of LAAS 13
1.3 Internship tools 14

2 Problematics, Objectives and State of the Art **15**
2.1 Preliminary notions 16
2.2 Problematics 17
2.3 Objectives 17
2.4 State of the art 18

3 Project Management **21**
3.1 Project plan 22
3.2 Work organisation 23
3.3 Risk management 23
3.4 General work approach and testing conditions 24

4 Possible Technical Solutions **27**
4.1 Using TSPs 28
4.2 Using a TSP to schedule the holes and Dijkstra's algorithm to choose the configurations 28
4.3 Using Generalised TSPs 29

5 Formalisation and Implementation **31**
5.1 Formalisation 32
5.2 Starting from an initial set of several base poses 35
5.2.1 Generation algorithm 35
5.2.2 First approach with exact costs 37
5.2.2.1 Building the cost matrix 37
5.2.2.2 Solver used 39
5.2.2.3 Results 40
5.2.3 Second approach with approximate costs 42
5.2.3.1 Reducing the costs' evaluation time 42

5.2.3.2 Results 43
5.2.4 Third approach: approximate costs and iterations 45
5.2.4.1 The iterative process 45
5.2.4.2 Results 46
5.2.5 Comparative results 47
5.2.6 Tackling bigger scale problems 49
5.3 Starting from an initial set of arm configurations 50
5.3.1 Introduction 50
5.3.2 General principle 52
5.3.3 Clustering using LAAS algorithm 54
5.3.4 Selecting the base poses found by the generation algorithm 55
5.3.4.1 Making the generation algorithm more efficient time-wise 56
5.3.4.2 Covering algorithm applied on the initial base poses sets 58
5.3.4.3 K-medoids and covering algorithm applied on the initial base poses sets 60
5.3.5 Comparative study 65
5.3.5.1 Comparative study on the 20 holes part 65
5.3.5.2 Comparative study on the 240 holes part 67
5.3.6 Study of the solving approach using Dijkstra's algorithm 69

6 Critical Viewpoint and Conclusions 71
6.1 Critical viewpoint 72
6.2 Perspectives 73
6.3 Personal conclusion 73

Bibliography 73

Appendix A Glossary 76

Appendix B Acronyms 77

Appendix C Deeper Algorithm Descriptions 78

List of Figures

1.1 TIAGo robot 12
1.2 Representation of the TIAGo robot along with a portion of the engine pylon in Gepetto Gui 14

2.1 Visualisation of different configurations in Gepetto-gui 16

3.1 Initial project plan 22
3.2 Visual representation of both parts in Gepetto Gui 25

5.1 Graph representing an instance of our problem 33
5.2 Visualisation of different configurations involved in the generation process 36
5.3 Overview of the base poses and the part 37
5.4 Average results of the method involving exact costs for different sizes of initial configuration sets 40
5.5 Further computation time analysis regarding the method involving exact costs . 41
5.6 Average results of the method involving approximate arm costs for different sizes of initial configuration sets 43
5.7 Average computation times of the method involving approximate arm costs for different sizes of initial configuration sets 44
5.8 Average computation times of the method involving iterations over approximate arm costs for different sizes of initial configuration sets 47
5.9 Comparison of average results of the 3 methods for different sizes of initial configuration sets 48
5.10 Time computation breakdown on a 240 holes part - approximate method 49
5.11 General principle of the approach involving the generation of additional arm configurations 53
5.12 Influence of the value of parameter N_{tries} in the clustering algorithm 55
5.13 Influence of the value of parameter K in the generation algorithm 57
5.14 Influence of the value of parameter N_{iter} in the Set Cover algorithm 59
5.15 Overview of the base poses clusters and the part 62
5.16 Overview of the base poses clusters and the part, with neighbours 63

List of Tables

5.1 Comparison between the initial approximate method and its extended variant . . 50
5.2 K-medoids algorithm results applied on an initial base poses set, for different values of $K_{medoids}$ and $N_{neighbours}$ 64
5.3 Comparison of all methods and their variants for a 20 holes problem 66
5.4 Comparison of all methods and their variants for a 240 holes problem 67
5.5 Comparison between the GTSP and Dijkstra algorithms 70

List of algorithms

1 Generate base poses (simplified) 35
2 Iterating over the approximate method 46
3 LAAS algorithm to find clusters 54
4 Base selection using multiple Set Cover algorithms and a GTSP 59

5 Generate base poses 79
6 Build the cost matrix 80
7 Build the approximate cost matrix 82
8 LAAS algorithm to find one cluster around a given hole 83

Chapter 1

Introduction

This chapter deals with the subject of this internship and links it to its context (section 1.1), describes the company (section 1.2) as well as the tools available to perform this internship (section 1.3).

1.1 Subject and context

The aim of this internship is to schedule the tasks of a robot as part of the ROB4FAM (Robots For the Future of Aircraft Manufacturing) common laboratory between Airbus and the LAAS (Laboratory of Analysis and Architecture of Systems). One of the main studies related to ROB4FAM is the use of robots to perform deburring tasks during the manufacturing of aircraft parts. While these tasks are mostly performed by machines already, they still require an operator to program them in order to tell them what to do. Using automated robots could therefore reduce the workload of operators by removing the need to program these machines. Several challenges appear as the robots will not only need to be autonomous and responsive to the way their environment changes, but will also have to work while humans are nearby and potentially interact with them.

More specifically, the use of a TIAGo robot developed by PAL Robotics is studied to perform deburring tasks on an A320 engine pylon. As shown in figure 1.1, the robot has a moving base with wheels, on top of which is attached a body that can move up and down and an arm connected to it that can move similarly to a human arm and hold items, such as a drill in our case. This robot can therefore be used to visit the given holes of an aircraft part and perform a deburring action on these.

Figure 1.1: TIAGo robot

The subject of the internship linked to this context is as follows: "Robotic task scheduling for industrial applications". The purpose of this internship is to develop effective task scheduling algorithms, as part of the ROB4FAM laboratory. Indeed, one of the main sub-problems related to the deburring of a given part to machine is the task scheduling. This internship thus focuses on this sub-problem and studies ways of scheduling the movements of the robot in order to

reach and deburr all the holes in the most effective way. The aim of the project being industrial applications, one of the key aspects is to reduce the time associated to the deburring process. Indeed, as the process will be repeated over and over again in the industry, the time needed to perform the said process is something that needs to be optimised, in order to be able to machine as many parts as possible in a given time window. Furthermore, the computation times also need to be kept as low as possible for similar reasons. We will therefore aim at developing effective solutions time wise, whether it be regarding the computation times or the time needed to actually perform the deburring task.

1.2 Brief description of LAAS

The internship is done within the Laboratory of Analysis and Architecture of Systems (LAAS), which is a laboratory of the French National Centre for Scientific Research (CNRS). The LAAS is mainly specialized in research activities related to information sciences and technologies. It deals with the study of complex entities of all kinds, ranging from robotics and integrated systems to biological ones. The associated domains are therefore as wide, with aeronautics, production, defense or healthcare belonging to these domains.

The LAAS consists of around 700 people, assigned to 6 research departments:

- Network and Critical Infomation Processing
- Robotics
- Decision and Optimisation
- Microwaves and Optics
- Energy Management
- MicroNanoBio Technologies

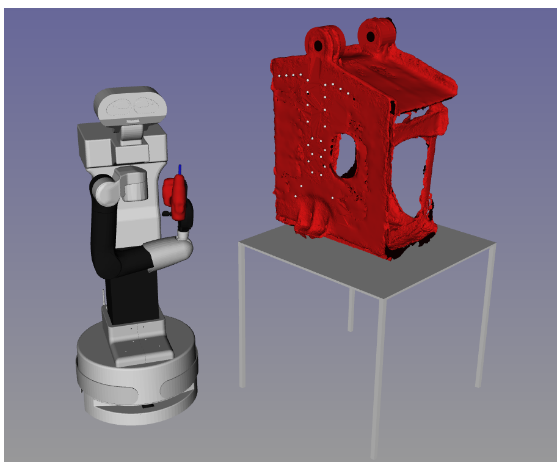
The internship takes place within two teams of these departments, more specifically the Gepetto team from the Robotics department and the ROC team from the Decision and Optimisation one:

- The Gepetto team on the one hand is specialized in the movements of anthropomorphic systems, especially regarding the generation and the analysis of these movements.
- The ROC team (Operations Research, Combinatorial Optimisation and Constraints) on the other hand, is specialized in the modelling and the solving of optimisation problems, whether they be combinatorial or discrete, and constraint satisfaction problems.

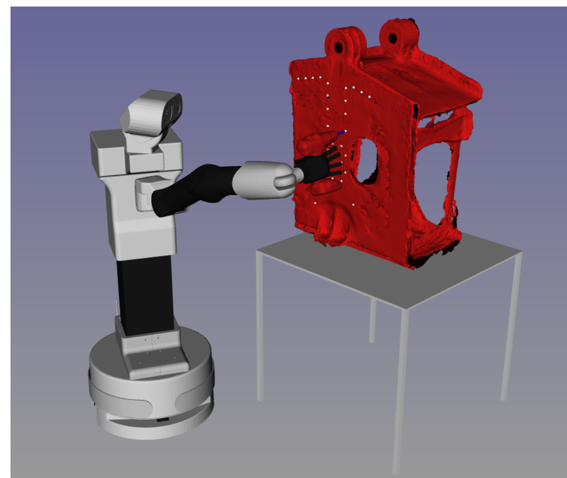
The internship thus takes part in between these two teams, as it not only requires solving a combinatorial optimisation problem, but also generating the movements of the robot to achieve the given task.

1.3 Internship tools

Several working tools were used during this internship. One of which being a software developed at LAAS. This software called HPP (Humanoid Path Planner) allows generating robot movements in order for them to manipulate objects in a given environment [1]. Movements are generated from an initial state to a final one and taking into account a set of constraints either related to the environment or the robot itself. This software consists of a C++ server which can be interacted with using Python commands. I therefore have access to a Python script which initializes the environment, mainly by loading a model of the robot and the part to deburr. This script also contains several methods and functions that can be used to generate the movements of the robot. Linked to these tools is a visualisation software, Gepetto Gui, also developed by LAAS as suggested by its name. Figure 1.2 below shows the TIAGo robot and the part to deburr as it is represented in Gepetto Gui. We can see the robot holding the drill in figure 1.2a with its arm retracted, and then holding its drill in front of one of the holes to deburr in figure 1.2b. This movement, along with all the positioning of objects and the introduction of several constraints have all been defined using a Python script interacting with HPP.



(a) TIAGo robot holding the drill with a retracted arm position



(b) TIAGo robot visiting one of the holes using its drill

Figure 1.2: Representation of the TIAGo robot along with a portion of the engine pylon in Gepetto Gui

The entirety of the internship is carried out using Python, and the results can easily be visualised through Gepetto Gui. More tools were used in order to actually deal will the problem and are introduced later in the report.

Chapter 2

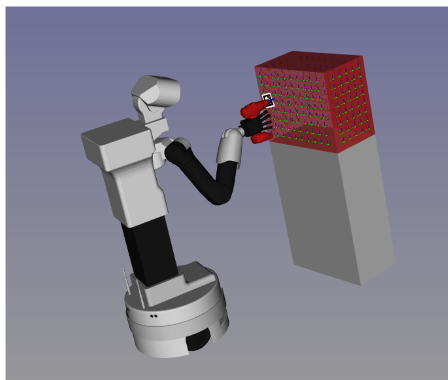
Problematics, Objectives and State of the Art

This section explains the related objectives (2.3) and problematics (2.2) and describes the state of the art related to the subject (2.4), as well as preliminary notions necessary to understand the upcoming parts of this report (2.1).

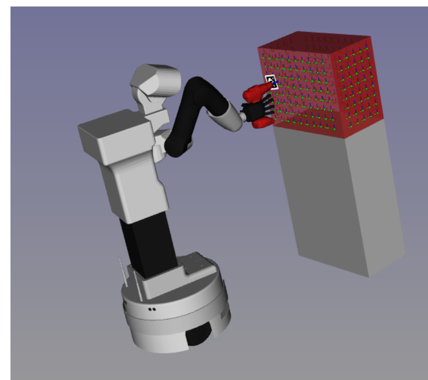
2.1 Preliminary notions

The aim is to define and illustrate some technical notions needed to understand the problem and the way we tackled it.

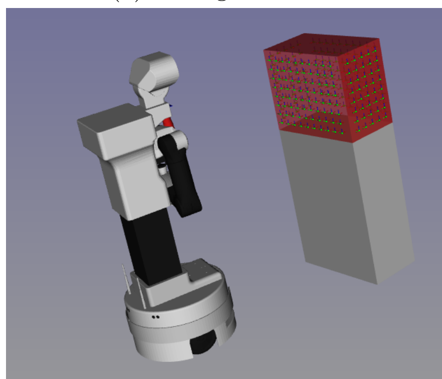
We denote *configuration* a vector containing the position and orientation of all the joints of our robot, at a given time. This can be seen as a snapshot of our robot, as it entirely describes its pose at a given time. Additionally, we will call a *base pose* the position and orientation of the base of the robot within a given configuration. Similarly, an *arm configuration* corresponds to the position and orientation of all the joints of the robot that are not a part of its base. As a result, a given configuration of the robot can be split up in 2 distinct parts, one of which telling us the general space location of the robot, the other describing the pose of its arm and upper body. Figure 2.1 below illustrates these notions.



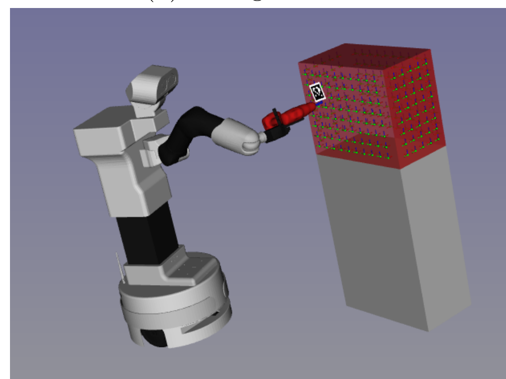
(a) Configuration A



(b) Configuration B



(c) Configuration C



(d) Configuration D

Figure 2.1: Visualisation of different configurations in Gepetto-gui

Configurations A, B and C all have the same base pose but different arm configurations. Indeed, while the position and orientation of the base of the robot is similar, the configuration of its

arms is not. Indeed, configurations A and B show the robot with its drill in a hole, but using a different arm configuration, while configuration C has its arm retracted. Finally, configuration D has a different base pose and arm configuration than all the other ones.

2.2 Problematics

As mentioned in section 1.1, the aim of the internship is to schedule the visit order of a given set of holes to deburr on a specific part. In order to perform these deburring tasks, the robot can move freely around the part in order to reach its holes. Indeed, as the base of the robot can translate and rotate on the ground, it has three degrees of freedom. Furthermore, the arm of the robot can also move freely thanks to several joints. As a result, the tip of the drill held by the robot has six degrees of freedom for a given fixed position of its base.

Therefore, each given hole can theoretically be reached by an infinite amount of robot configurations. Indeed, both the base and the arm of the robot have multiple ways of reaching a hole. The key element is thus to associate the right configuration to each hole. As mentioned above, the main problematic linked to this statement is the fact that each hole can be reached by an infinite amount of robot configurations.

Also, the parts we are studying can have holes on several of their sides. As a result, it is not guaranteed that the robot will be able to reach all the holes from the same base pose. Indeed, the purpose of this study is to be able to machine a part that requires the robot to move around it.

Several main problematics are thus appearing:

- The base of the robot needs to move in order to reach all the holes
- A given hole can be reached by an infinite amount of base poses
- A base pose associated to a given hole can reach it with an infinite amount of arm configurations

These are the main initial problematics, more specific ones are discovered later during the internship, mainly regarding computation time.

2.3 Objectives

With the problematics clearly defined, the objectives of the internship can be set. The aim is to develop a task scheduling algorithm which returns a tour of the holes to deburr while taking into account the large amount of configurations that can reach each given hole of a part and choosing the best configuration among these.

Several axes of optimisation are thus appearing: the order in respect to which each hole will be visited, as well as which base poses and arm configurations will be used. In other words, we do not only have to find in which order the holes should be visited, but also how they will be visited. Both aspects are dependent with each other. Indeed, the configurations used to reach a

set of holes have a direct influence on which order these holes will be deburred, as the movement between configurations can be faster or slower depending on how we schedule the holes' visiting order. Additionally, the amount of base poses used to complete the deburring task can also be optimised: it is indeed costly to use multiple base poses as the movements between them are more time-consuming. As an example, it is faster to deburr a given set of holes from a single base pose than having to switch to another one half-way, as this transition forces the arm to retract itself to its resting position, before having the robot move to its next base pose and unfold the arm to start the deburring process on the remaining holes. Furthermore, the use of multiple bases is associated with instability issues, as each movement of the base could lead to the robot getting lost.

We thus not only have to optimise the tour of the holes, but also the configurations used to visit them, as well as keeping the amount of base poses as low as possible in order to fulfil our main criterion, which is the total time needed to perform the deburring task.

The aim of the study is to find the shortest path to visit the holes of a part, by taking into account the optimisation axes mentioned above and keeping the computation time as low as possible.

2.4 State of the art

While this problem may seem similar to the well-known Travelling Salesman Problem (TSP), it is actually different because we do not only need to compute the shortest path visiting all the holes, but also take into account the fact that several configurations can reach each of the holes. If only one configuration could do so, the problem would indeed be a TSP.

We firstly had a look at several studies aimed at solving different variants of TSPs, in order to see what was already done in this field. As the TSP is a widely known problem with a lot of related variants, a major amount of research has already been done regarding these problems. One of these studies comes close to what we aim at doing, as it deals with two subnetworks of a delivery process, using lorries for high level tours which serve as mobile depots for a set of drones which do the deliveries [2]. This problem is close to our study, since we could consider the moving base of our robot as a lorry, and its arm as a drone tasked with visiting its customers (which would be the holes). However, our problem is not as complicated since we would only use one lorry and one drone, while the study considers several of these. We therefore realised that our problem did not need a dedicated optimisation approach that we would need to develop ourselves, as much more complicated problems have already been tackled and solutions exist for these. Instead, the goal is to find which algorithms to use and how to pair them with our robotic oriented tools, in order to get the best results. For this reason, we also studied research papers related to robotic task scheduling.

LAAS already developed a solution for this specific problem, but it needs to be improved. The algorithm, developed by Joseph Mirabel, splits the holes into clusters that can be reached from one single base pose, which each hole being associated to a single arm configuration. Then, multiple TSPs are applied: one to schedule the movement of the base poses, and one for each cluster, in order to schedule the movements of the arm and visit the holes. With the TSP algorithms being implemented using dynamic programming, the computation times were far

too high. Additionally, a single base pose was considered for each cluster of holes and only a single arm configuration was associated with each hole. In other words, the possible infinite amount of configurations able to reach a given hole was not explored, as a single configuration was associated to each of the holes. As a result, the tour was not only far from being optimal but also took too long to be computed, due to the TSP algorithm used.

We did not find studies involving a mobile robot having to machine a part (or perform similar tasks). A survey of the different approaches regarding the task scheduling problems [3] in robotics did not mention such a problem either, although some of the articles it describes deal with the choice of the base pose of the robot, the use of several different base poses to achieve a task is never mentioned.

The closest study to our problem was described in [4]. This article deals with a similar problem, where a manipulator arm fixed on the ground has to visit the holes of an industrial part, using a drill attached to its arm. The tip of the drill also has six degrees of freedom in this study. The method used to solve their scheduling problem is as follows (vastly simplified):

- First, a tour of the holes is computed using a TSP.
- Then, with several arm configurations being generated for each hole, a Dijkstra algorithm chooses the best ones, while following the tour initially defined.

This method can supposedly not be adapted to our study. Indeed, the fact that the tour is being computed initially represents a consequent drawback if we consider a moving base (see 4.2).

While the existing research regarding our problem is scarce, a colossal amount of optimisation algorithms have already been studied and developed. These can be associated with the tools developed at LAAS in order to solve our problem.

[INTENTIONALLY LEFT BLANK]

Chapter 3

Project Management

This section deals with the management of this internship, where we describe the initial project plan and its evolutions (3.1). We also provide the general approach used to tackle the problem (3.4), as well as the work organisation (3.2). Additionally, we describe and mitigate the different risks related to this internship (3.3).

3.1 Project plan

Figure 3.1 below presents the initial project plan. It was split up in 4 parts:

- The first familiarisation with the subject
- A first approach aimed at solving the problem by focusing on the base poses
- An improvement of the said method by enhancing the path planning algorithms used at LAAS, in order to get smoother movements of the arm
- The management of deliverables, whether they are for ENAC of for LAAS.

The two main parts of the internships were therefore supposed to be evenly split up, with the first one being planned to last longer due to a necessary familiarisation time and the implementation of algorithms possibly reusable in part 2. Although we well followed this initial planning, the second part was done in a different way than initially planned. Indeed, instead of finding ways to improve the path planning algorithms developed at LAAS, we tried to improve our results by combining some of the algorithms developed in part 1 with other methods, in order to get better movements regarding the arm. Therefore, our goal remained the same but the way to reach it changed completely. We needed to drastically improve the computation times of the methods implemented in part 1 in order for our algorithms to be a suitable solution to big scale problems.

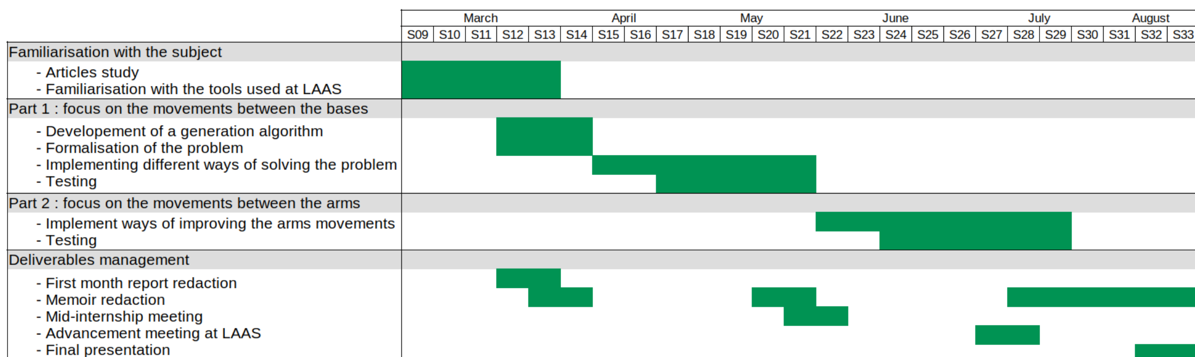


Figure 3.1: Initial project plan

3.2 Work organisation

This internship was carried out entirely via teleworking, mainly because of COVID-19 pandemic restrictions and capacity issues at LAAS. Moreover, no physical tools were needed to do the internship, as the testing could be done using a LAAS tool (Gepetto-Gui).

Several people at LAAS helped me achieve my End-of-Study Project, including Florent Lamiraux, my internship supervisor, who is a director of research from the Gepetto team and has worked himself on the movements generating software I am using. Moreover, Christian Artigues (director of research from the ROC team) and Cyril Briand (professor at UT3 - Paul Sabatier and researcher from the ROC team) can also help me, mainly in the optimisation field. In addition to this, several other researchers or PhD students from LAAS could help me if I need advice in more specific fields.

Additionally, Nicolas Barnier (teacher and researcher from the OPTIM team at ENAC) can also help me and surveys my progress regarding internship, as he is my ENAC tutor.

I take part in two meetings at LAAS every week. The first of which is held up on Monday as part of the ROB4FAM project, everyone who works for this project at LAAS is expected to attend the meeting, as well as an Airbus representative. The aim of the meeting is to have everyone describe their progress by going through what they did during the previous week as well as what they are planning to do in the upcoming week. Another meeting is organized every Tuesday, this one being specifically focused on the internship itself as Florent Lamiraux, Christian Artigues, Cyril Briand and myself attend it every week. Its goal is not only to assess my weekly progress and discuss the tasks I should do next, but it also serves as a much-needed link between the Gepetto and ROC teams, ensuring the effective conduct of the End-of-Study Project. This meeting also serves as a brainstorming time, where we all discuss possible solutions or improvements regarding the already implemented ones.

3.3 Risk management

Three main risks linked to this internship were noticed:

1. Experiencing difficulties related to programming: as programming is the main tool I will use in order to fulfil the objectives of this internship, it is therefore a critical part of the internship that could slow things down a lot if I come across issues. Mitigation: luckily, I have a solid experience related to coding thanks to my studies at ENAC and can thus refer to my courses if I ever need to. However, if I were to remain stuck, I could still look for solutions on the Internet or ask to my supervisor for advice.
2. Experiencing difficulties related to optimisation: optimisation is another key point of the internship that will be used to solve the problem. Failure to understand, create and apply optimisation strategies will most likely lead to the failure of the internship itself. Mitigation: Hopefully, meetings are organized every week with two optimisation experts that will guide me and help me understand the concepts. They are also available outside of the meetings if I ever need help.

3. Experiencing difficulties related to teleworking: doing an internship from home can be frustrating and hard to perform. It may also slow down the whole internship. Mitigation: thankfully, I already have some teleworking experience as I did my whole second year internship that way. The environment I am working in is also calm and welcoming and will therefore have no negative side effects on the effectiveness of my work. As mentioned previously, several meetings are organised every week to assess my progress. More meetings also take place with my ENAC tutor. Finally, all the persons I interact with are always available to help me overcome the potential difficulties related to teleworking.

No additional risks were found during the advancement of the internship. I however experienced some difficulties related to programming at the beginning of the internship, mainly linked to the way of handling LAAS manipulation and path planning tools. My internship supervisor, who worked on these tools himself, was always able to help me deal with the issues I encountered. As for the optimisation, no issues were encountered at all, since we used already developed algorithms to deal with this aspect. However, some difficulties linked to teleworking appeared, even though we thought of mitigation measures. The main issues encountered were technical issues. Indeed, I encountered several memory issues on my computer because some specific instances of LAAS algorithms I had to use massively required too much memory and eventually led to crashes. Because of this, some of the testing could not be carried out properly. As a result, Florent and I found alternative ways of dealing with the problem, which eventually cost us some time.

3.4 General work approach and testing conditions

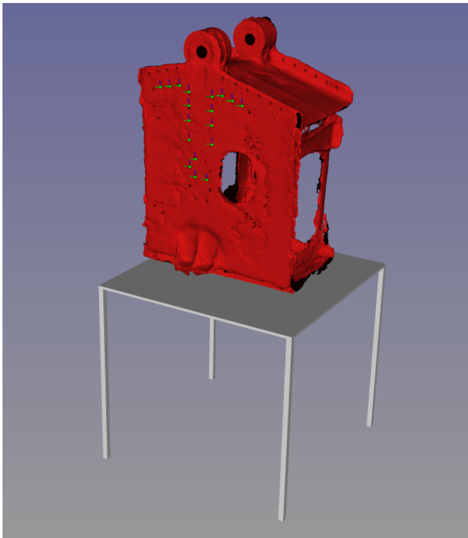
The general approach to tackle this internship is close to an incremental approach, where the solution is built step by step. Indeed, we developed, implemented and tested several main functions one after each other, by analysing the strengths and weaknesses of the former. This approach helps to improve our implemented solutions very easily, as we can combine the noticed strengths while trying to either avoid the eventual pitfalls we come across or find ways to mitigate their effect. This way of handling the problem is necessary to build the best possible solution as we explore several possibilities of improvement.

Testing therefore plays an important role in the advancement of our project, as it is a much necessary step to assess the performance of each method and thus know which direction to take for the next increment. A lot of randomness is used in our studies. Indeed, the algorithms developed by LAAS, whether they focus on the generation or the path planning, are based on random methods. For this reason, we cannot simply test a given method once, but have to repeat the same tests multiple times and compare the average results. Each test will be repeated 10 times for this matter.

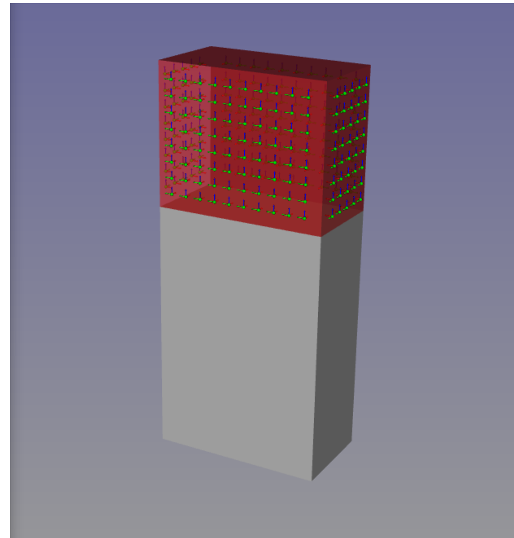
As for the testing environment, the part provided by Airbus is not suitable for our study. Indeed, it is only a fraction of an engine pylon and has 18 holes to deburr. This is not enough for us to evaluate the time effectiveness of our algorithm, which should be able to tackle bigger problems. Additionally, we noticed that our robot can easily reach all the holes without having to move its base, which simplifies the problem a lot as we initially aimed at solving problems requiring the robot to move around. For these reasons, we asked Airbus to share with us the model of the

whole part, which they could not do. Consequently, we had to develop our own part in order to properly test our solutions. The part is a cuboid with 240 holes scattered around 4 of its faces. This allows us to have a big instance to deal with, as well as the need to use several base poses in order to reach all the holes. We split up our testing in two parts. First, we only consider 20 of the holes (10 on one side, and 10 on the other), in order to do a preliminary assessment of a given method. Then, we consider the whole part if we think that the method is suitable for such a big scale problem.

Figure 3.2 shows both parts previously mentioned, were the holes are represented by their frame basis in green and blue.



(a) Model of the part provided by Airbus



(b) Model of the cuboid we developed

Figure 3.2: Visual representation of both parts in Gepetto Gui

[INTENTIONALLY LEFT BLANK]

Chapter 4

Possible Technical Solutions

This section presents various technical solutions which could help solve the given problem.

4.1 Using TSPs

This method, which is the one already implemented at LAAS works very simply: a clustering algorithm generates and associates a base pose with a set of holes (a base pose is associated to a hole via an arm configuration), and keeps adding base poses until all the holes are covered. Then, a TSP is used to schedule the movement between the base poses. Finally, one TSP is applied per base pose, in order to schedule the movements of the arms.

This method has a major drawback as the configurations of the robot are never optimised. Instead, a single random arm configuration is generated and associated to each hole. Similarly, no more base poses are generated than what is needed to cover all the holes. As a result, there is no possible choice to do regarding these configurations. This will have a strong influence on the length of the final tour because while the TSPs will create a nearly optimal tour, this tour is only nearly optimal for the configurations considered. Indeed, other configurations could yield better results, as there is an infinite amount of them in theory.

Regarding computation times, it is clear that this method can be very time effective, as it uses several TSPs on small problems. While the TSP implemented at LAAS was slow, many other methods involving heuristics could be used in order to reduce the computation time.

This method can therefore solve our problem in a relatively short time, but with poor results.

4.2 Using a TSP to schedule the holes and Dijkstra's algorithm to choose the configurations

This is the approach described in [4]. This method starts by scheduling the holes to visit, without taking the robot into account. Indeed, a TSP is applied between the holes, optimising the total metric distance between them. Then, several configurations are generated for each hole, and Dijkstra's algorithm is used to choose the best configurations by finding the shortest path following the initial scheduling. This approach allows to find the best sequence of configurations to go through, within a previously planned visit order of the holes.

This method cannot be directly implemented to solve our problem. Indeed, while the initial scheduling of the holes can be an acceptable solution for a fixed base, it is not effective if the base can move. Indeed, if holes h_1, h_2 and h_3 have to follow each other according to the initial scheduling but h_2 cannot be reached by the same base pose as h_1 and h_3, the robot will have to reach h_1, then move its base and reach h_2, before moving again to its initial base pose in order to reach h_3. This example requires to unnecessarily move the base in order to reach all the holes. Indeed, as the initial scheduling of the holes does not take into account the base of the robot, it cannot be applied to our problem.

However, this method has several advantages. First of all, several configurations are associated to each hole and choices are made in order to select the best ones. The computation times are also very low, as only one TSP is needed, followed by a Dijkstra.

This method could be associated with the one mentioned above: indeed, once the clustering is done, a TSP could be applied to each cluster of holes, followed by a Dijkstra. This would fix the issue regarding unnecessary base movements, as the method could be applied within each single cluster. Computation times would be slightly higher, as both a TSP and a Dijkstra algorithm are needed for each cluster. However, since a choice is made between the arm configurations, this method could provide better results than the former one.

Nevertheless, the initial scheduling of the holes could still be an issue. Indeed, even if the Dijkstra algorithm will choose the best arm configurations, these are only the best arm configurations corresponding to the chosen visit order of the holes. Since this visit order does not take into account the arm configurations, it is entirely possible that another shorter tour regarding the holes could give better results regarding the deburring time.

With several advantages and drawbacks, this method, applied to each cluster, will be compared with our solutions in order to see which one stands out.

4.3 Using Generalised TSPs

A Generalised Travelling Salesman Problem is a TSP where several sets of nodes have to be visited, but only one node per set has to be chosen. The aim of the GTSP is therefore to find the best tour linking the best nodes chosen in the sets. This corresponds to our needs, as we want to find the shortest tour between the holes, while choosing the best configuration to associate to each hole.

Using a single GTSP to schedule the visit order of all the holes while choosing the best configuration could be very time-consuming. Indeed, this could correspond to an instance of a $\mathcal{NP}$-Hard problem with thousands of nodes. Thankfully, heuristics have been implemented to solve GTSPs efficiently [5]. Furthermore, we can find ways to reduce the size of the problem if we need to do so.

The main advantage of using one or several GTSP is that it takes into account both the choices of configurations and the scheduling, while the two other methods described above either focus on the scheduling or on the choices, but never both at the same time. As a result, the final tour found using a GTSP should be better than the two other methods. The main drawback will of course be the computation time, as this method takes into account all the parameters simultaneously, in order to increase the quality of the output.

With the advantage described above, a resolution based around GTSPs seems to be the most adapted to our study, if the computation times are feasible. Therefore, we will find different ways to solve our problem as one or several GTSPs, and compare the advantages and drawbacks of the implemented variants in order to find out which ones are the most suitable.

While this solution appears to be the most suitable one, the need to describe our problem with a cost matrix also happens to be a problem by itself, since we have to find ways of evaluating the cost linked to the movement between two configurations. This cost evaluation process can indeed be costly time-wise (see section 5.2.2).

[INTENTIONALLY LEFT BLANK]

Chapter 5

Formalisation and Implementation

This section presents a formalisation of our problem (5.1), as well as all the methods implemented and their variants. Two major approaches were studied: section 5.2 deals with different methods involving a large set of initial base poses, while section 5.3 describes methods involving a large set of arm configurations, starting from a small set of base poses.

5.1 Formalisation

As mentioned in 2.2, we can associate several configurations to each hole of the part. At this point, it does not matter whether these configurations have a common base pose or not, the aim of this section being to mathematically illustrate our problem. However, the specificities of these configurations will matter later on, during the actual solving of the problem.

Let us define H and Q, respectively the set of holes and the set of configurations. Let $G(V, E)$ be a graph where $V = \{0, 1, 2, ..., n\}$ is the set of nodes and where $E = \{(i, j), i, j \in V, i \neq j\}$ is the set of edges between these nodes. A given node i is represented by

$$\begin{cases} i : (h_k, q_l) \text{ with } h_k \in H \text{ and } q_l \in Q \text{ such that } h_k \text{ can be reached by } q_l & \text{if } i \neq 0, \\ i : (\emptyset, q_0) \text{ with } q_0 \text{ being the initial configuration} & \text{if } i = 0 \end{cases}$$

Here we consider that the initial configuration q_0 cannot reach any holes. Indeed, this configuration corresponds to where the robot is before starting to perform its deburring task. It is highly possible that the robot could be initially stored somewhere for example. This node corresponds to the depot node commonly described in TSP problems.

Let $V_0, V_1, V_2, ..., V_K$ be a partition of V into $|H| + 1$ disjoint subsets such that $\bigsqcup_{k=0}^{K} V_k = V$. Each subset V_k =$\{(h_k, q_l), q_l \in Q\}, \forall k > 0$ contains all the configurations that can reach the kth hole while V_0 contains the initial node.

Figure 5.1 shows the graph associated with the following instance of our problem:

- $H = \{h_1, h_2, h_3, h_4\}$;
- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$;
- We assume that:
  - h_1 can be reached by q_1, and q_2. As a result:

$$(h_1, q_1) \in V_1$$
$$(h_1, q_2) \in V_1$$

  - h_2 can be reached by q_3:

$$(h_2, q_3) \in V_2$$

  - h_3 can be reached by q_4, q_5 and q_6

$$(h_3, q_4) \in V_3$$
$$(h_3, q_5) \in V_3$$
$$(h_3, q_6) \in V_3$$

- h_4 can be reached by q_7 and q_8

$$(h_4, q_7) \in V_4$$
$$(h_4, q_8) \in V_4$$

The greyed areas represent the subsets from which a single node must be chosen. Indeed, the goal is to find the best tour allowing to visit all the holes by associating one configuration to each one of them. This corresponds to a GTSP which we formalise below.

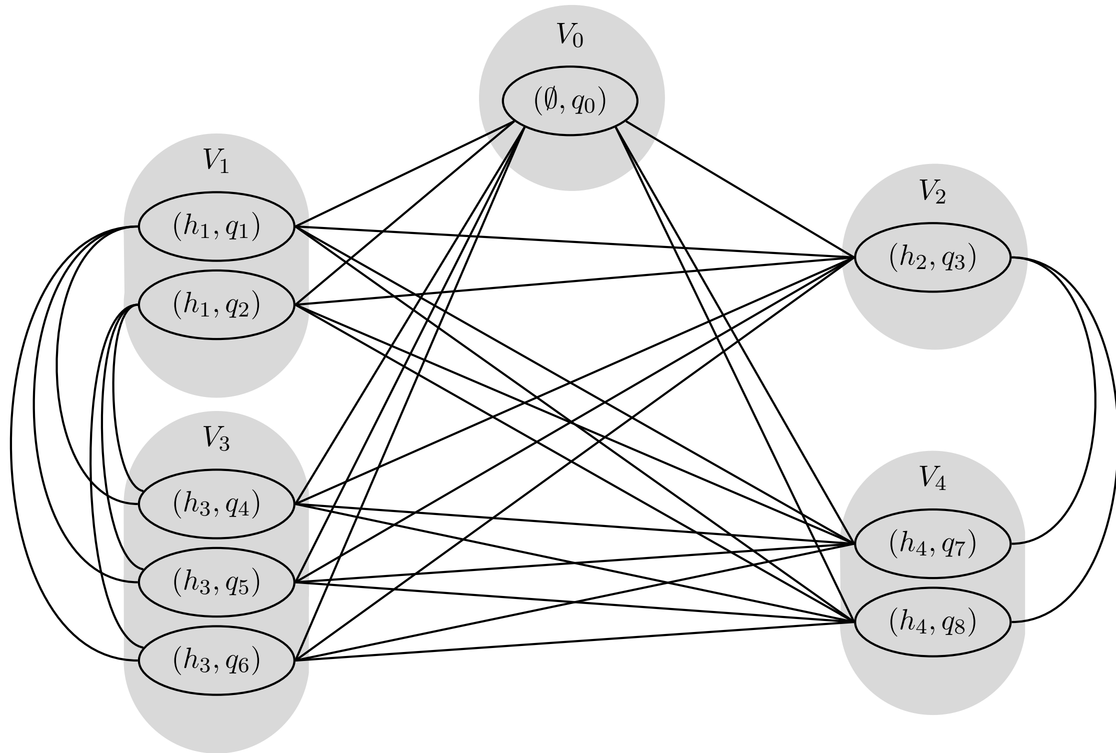


Figure 5.1: Graph representing an instance of our problem

Let c_{ij} represent the cost between nodes $i : (h_k, q_l)$ and $j : (h_{k'}, q_{l'})$, which corresponds to the cost it takes to go from the robot deburring hole h_k in configuration q_l to the robot deburring hole $h_{k'}$ in configuration $q_{l'}$. *Note*: q_l and $q_{l'}$ can or cannot have the same base pose, as a result, we will have to create a cost which models this possibility (see 5.2.2.1).

As each hole only needs to be visited once by the drill, it is straightforward to understand that only one node needs to be chosen in each subset. Let x_{ij} be the decision variable so that:

If $i \neq j$:

$$x_{ij} = \begin{cases} 1 & \text{if the route goes from node } i \text{ to node } j \\ 0 & \text{otherwise.} \end{cases}$$

If $i = j$:

$$x_{ii} = \begin{cases} 1 & \text{if node } i \text{ is } \textit{not} \text{ chosen in the optimal solution} \\ 0 & \text{otherwise.} \end{cases}$$

The GTSP can be formalised as follows (adapted from [6]):

$$\min \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} c_{ij} x_{ij} \tag{5.1}$$

subject to:

$$\sum_{i \in V} x_{im} = \sum_{j \in V} x_{mj} = 1 \qquad \forall m \in V \tag{5.2}$$

$$\sum_{i \in V_k} x_{ii} = |V_k| - 1 \qquad k = 0, ..., K \tag{5.3}$$

$$\sum_{i \in T} \sum_{\substack{j \in T \\ j \neq i}} x_{ij} \leq |T| - 1 \qquad \forall T \subseteq V \text{ with } T \cap V_k = \emptyset \text{ for at least one but not all } k \tag{5.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in V \tag{5.5}$$

The objective function (5.1) means that the goal is to minimize the cost associated to the chosen tour. Constraint (5.2) ensures that a given node either does not get visited at all or only gets visited once, with a single edge leading to it and a single edge coming out of it. Constraint (5.3) ensures that exactly one node is chosen in every subset. Constraint (5.4) is the subtour elimination constraint, ensuring the solution will have only one tour. Constraint (5.5) is the domain constraint on the decision variable.

With the problem now formalised as a GTSP, we can study different ways of generating the set of configurations to choose from. There are different ways of generating an initial set of configurations. Indeed, we can in theory associate an infinite amount of base poses or arm configurations to each hole. However, it is not suitable to generate such a high amount of configurations because of obvious memory and computation time constraints. Instead, we either focus on a large initial set of base poses, or arm configurations. Moreover, the time needed to evaluate a given cost c_{ij} can be high, as it is linked to the movement between two configurations.

5.2 Starting from an initial set of several base poses

This section describes our first approach which corresponds to feeding the GTSP with a large initial set of base poses. 5.2.1 presents the algorithm used to generate such a set of base poses, 5.2.2, 5.2.3 and 5.2.4 present three ways of tackling the problem using this initial set of base poses and 5.2.5 shows a comparison of these methods.

5.2.1 Generation algorithm

A set of configurations has to be generated in order to further select the best ones. In order to do this, we use generation tools developed at LAAS. The process is summarized in algorithm 1, where we compute N random configurations able to reach each of the holes, before checking if all the other holes can be reached from all the base poses of the configurations we computed. *Note:* We denote $D^{keys:values}$ a dictionary associating a set of *keys* to their respective *values*.

Algorithm 1: Generate base poses (simplified)

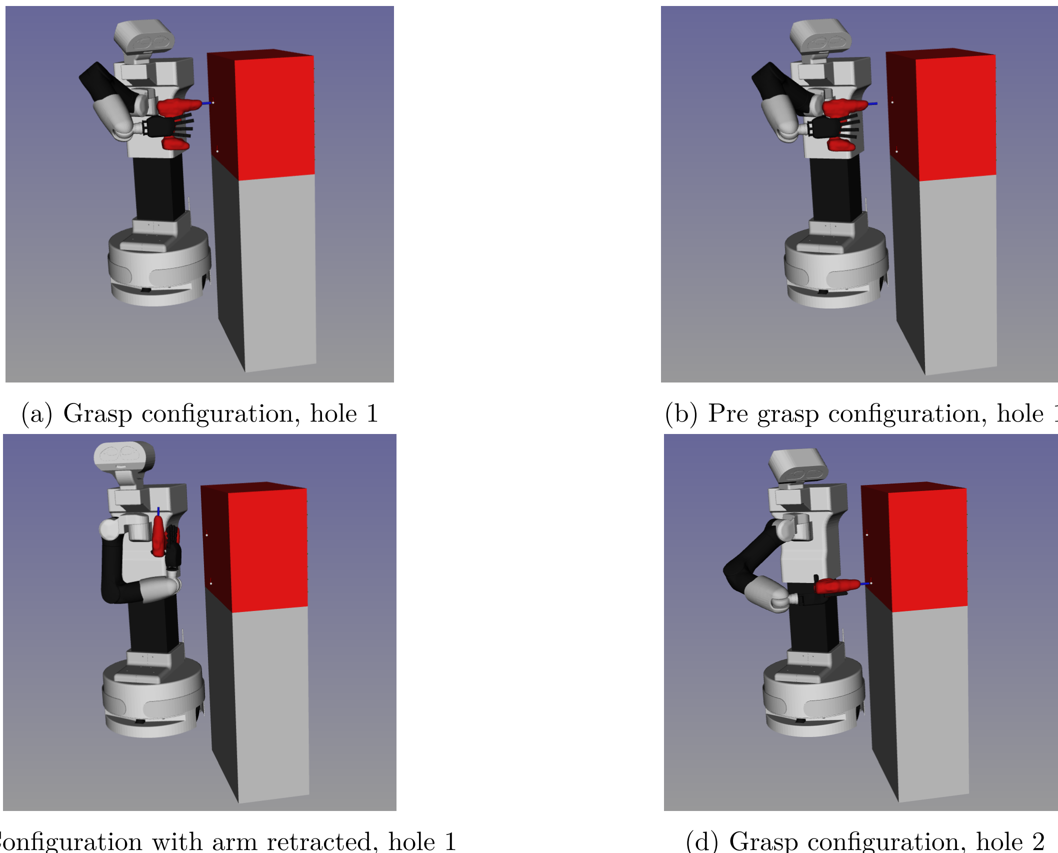
Input: $N, holes$
Output: $D^{holes:configs}, D^{configs:holes}$
1 **Initialize:** $D^{holes:configs} \leftarrow \emptyset, D^{configs:holes} \leftarrow \emptyset$
2 $D^{holes:configs}, D^{configs:holes} \leftarrow genInitialSet(N, holes)$
3 $D^{holes:configs}, D^{configs:holes} \leftarrow lookForReachableHoles(D^{holes:configs}, D^{configs:holes})$
4 **return** $D^{holes:configs}, D^{configs:holes}$

This algorithm splits up into two parts. Firstly, several robot configurations reaching each hole are computed (line 2). We actually associate and store 3 subconfigurations each time, which are necessary to well define how the robot can reach a hole. Indeed, we consider the initial configuration of the deburring task, where the drill is near the hole, as well as the final configuration of the deburring task which corresponds to the drill being inside the hole. These configurations are respectively called the *pre grasp* and the *grasp* configurations in HPP. Both configurations help to define the deburring process, which we represent by a translation. Therefore, the difference between both configurations is a simple translation of the drill. Finally, we also store the configuration with the arm of the robot retracted in its resting position, which will be useful when considering the movements of the base of the robot, done with the arm retracted. All 3 subconfigurations obviously have the same base pose by design.

Secondly, we check whether each computed base pose can reach each of the other holes of the part by only moving its arm (line 3). We therefore try to generate additional configurations by keeping the same base poses but only changing the configuration of the arm, in order for each base pose to be able to reach several holes. Each base pose will only be able to reach each of its reachable holes with a single arm configuration, as opposed as section 5.3 where more arm configurations are generated but with a lower amount of base poses. The results are stored in two dictionaries, the first one associating each hole with the base poses that can reach it; the other one associating each base pose with the holes it can reach. These two dictionaries define the graph corresponding to the instance of our problem. Figure 5.2 below illustrates the configurations mentioned, where a grasp configuration (figure 5.2a), a pre grasp configuration (figure 5.2b), and a configuration with the arm retracted (figure 5.2c) associated to a hole are

shown. Also, we show a grasp configuration reaching another hole from the same base pose (figure 5.2d), illustrating the second part of the generation process.

A more in depth explanation of how this generation algorithm works is presented in Appendix C.1.



(a) Grasp configuration, hole 1

(b) Pre grasp configuration, hole 1

(c) Configuration with arm retracted, hole 1

(d) Grasp configuration, hole 2

Figure 5.2: Visualisation of different configurations involved in the generation process

Figure 5.3 below shows how the base poses are scattered around the part. 240 holes are considered, for each of which one configuration was generated. This figure corresponds to a view of the scene as seen from above. The part is represented by a black rectangle. As for the base poses, each one is represented by a red circle and an arrow, pointing towards where the body of the robot is facing. Furthermore, the depot is at the bottom of the figure, represented by a black cross (barely visible here). We can see that the base poses are well scattered around the part. Furthermore, we have seen that the base poses do not necessarily need to face the part in order to reach its holes. Also, due to its location, we could consider the depot itself as one of the candidates to reach the holes, as it is close to other base poses. *Note:* For the sake of readability, the arm configurations are not represented on this figure, but it is important to remind that each of the base poses represented can reach a given amount of holes, with one arm configuration being associated to each of these holes.

We built an algorithm able to generate $N \times N_{holes}$ base poses, where N_{holes} is the number of holes, which can all reach a given amount of holes. Even though each base pose can only reach a given hole with a single arm configuration, multiple base poses can theoretically reach the said

hole. We therefore have several ways of deburring every hole, and can now focus on solving the problem from this set of configurations.

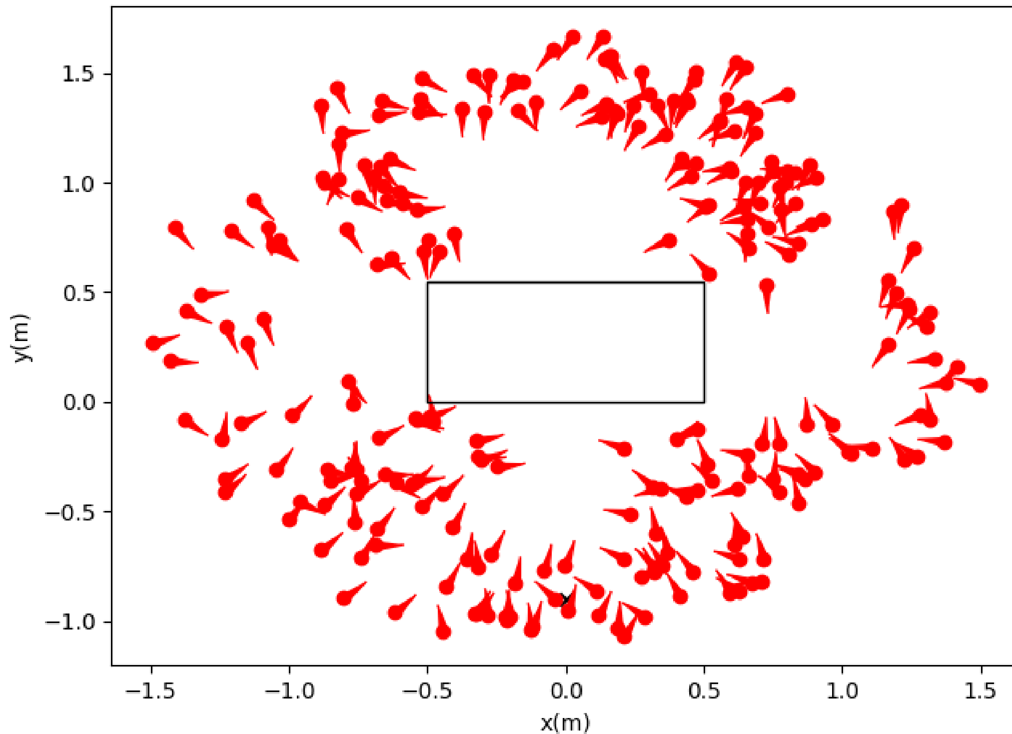


Figure 5.3: Overview of the base poses and the part

5.2.2 First approach with exact costs

This section describes the very first way implemented to solve our problem as well as several notions that will be re-used to build other methods. First, we explain how to evaluate the costs between the configurations previously generated (5.2.2.1), then, we choose a GTSP solver (5.2.2.2). Finally, we test this method (5.2.2.3).

5.2.2.1 Building the cost matrix

In order to solve the problem as a GTSP, it is necessary to associate a cost matrix to our problem. The idea presented here is to use the most realistic costs as possible.

While we already have a set of nodes thanks to the configurations generated, the goal is now to associate a cost to every edge linking these nodes. Thankfully, the Gepetto team, specialized in motion planning has already developed path planning algorithms, which allow to compute a path between two given configurations, while taking into account several constraints. The main constraints are related to the collisions and therefore allow to generate a feasible path between an initial and a final state. A generated path has a length attribute which corresponds to the distance between two configurations in the configuration space, along the said path. We can use

this length to accurately represent the cost between two nodes. Indeed, the longer the path, the longer it will take to execute the movement, and the higher the cost will be.

With a first evaluation of the cost, some assumptions have to be made (*which will remain unchanged during the whole study*):

- We do not take gravity into account: in other words, the cost associated to a path between node n_1 and node n_2 will be the same as between node n_2 and node n_1. Two reasons lie being this assumption: first of all, the path planning algorithms developed by LAAS do not differentiate the lengths associated to a path and its reversed instance. Then, taking gravity into account could lead to an unnecessarily complex cost matrix which would lead to longer computation times regarding the solver. Indeed, it is faster to solve symmetrical problems. Thus, the cost associated to a movement between two given nodes n_1 and n_2 will be:

$$cost_{n_1 \rightarrow n_2} = cost_{n_2 \rightarrow n_1} \tag{5.6}$$

- We do not evaluate the translation between the pre-grasp and the grasp state. Indeed, this transition has proven to be of a very similar cost every time it happens and will have almost no impact on the tour computed. Furthermore, this transition does not take into account the efforts applied on the drill, as this could be the point of another study. *Note*: if we could model the efforts associated to a deburring action into a cost, then the cost may indeed vary depending on which configuration the robot is in, and it should be taken into account when computing the tour.

- The cost associated to movements between the base poses have to be weighted. Indeed, the length attribute of a given path only represents a distance between configurations. However, this distance has a poor meaning when comparing two different types of movements. As the base moves slower than the arm in reality, we have to weigh the base distances in order for them to match with the arm distances. According to Florent Lamiraux, we can suppose that the base moves 2.5 times slower than the arm (with an estimated $20cm/s$ and $50cm/s$ respectively). Furthermore, each movement of the base is associated with a probability for the robot to get lost, which means it would have to re-calibrate in order to retrieve its planned path. As a result, we will add a constant K_p to the cost every time the base needs to move. Therefore, the cost associated to a movement of the base between two base poses q_1 and q_2 will be:

$$cost_{q_1 \rightarrow q_2} = length_{q_1 \rightarrow q_2} \times 2.5 + K_p \tag{5.7}$$

  The cost between two arm configurations q_1 and q_2 which do not require to move the base remains:

$$cost_{q_1 \rightarrow q_2} = length_{q_1 \rightarrow q_2} \tag{5.8}$$

  Note: it is not needed to further weight the movements of the base in order to take into account the fact that we want to choose as few base poses as possible. Indeed, the cost associated to a movement between two holes that require switching base poses will be naturally higher than if the same movement could be done from the same base pose. The first case would require the robot to fully retract its arm from the first hole to its resting

configuration within the first base pose, then move towards the second base pose before extending its arm again in order to reach the second hole, whereas a single movement of the arm would be needed in the second case, thus corresponding to a much lower cost.

After an assessment of the typical lengths associated to the base's movements, the value of K_p is set to 10, in order to have a sufficient impact on the cost, while not accounting for too much of it. Also, we do not add K_p when considering movements involving the depot, as these are necessary and cannot be avoided.

Thanks to these assumptions, we can now associate a cost to the computed path between two given nodes, which properly takes into account the movements of the base of the robot and of its arm. As a result, if the movement between two given nodes does not need to move the base, we will apply the cost defined in Equation (5.8). Similarly, if the base needs to move, its movements will be associated to the cost defined in Equation (5.7), while the movements of the arms will still be linked to the cost defined in Equation (5.8).

The costs between all the nodes are then stored in a matrix. As the computation of the paths is necessary to get the costs, the paths are also stored in a matrix. Once the problem is solved, the paths will be taken from this matrix in order to build the tour. An in-depth description of the matrix computation algorithm can be found in appendix C.2.

5.2.2.2 Solver used

With the cost matrix computed, we can now feed it to a GTSP solver in order to solve our problem. The ROC team advised me to use GLNS, which is a Large Neighbourhood Search Heuristic developed at the University of Waterloo [5] and aimed at solving GTSPs. According to their study, this algorithm gives similar results as other existing solvers in regular problems, as well as better results in more specific ones. This solver is implemented in the Julia language, and the code is freely available at `https://github.com/stephenlsmith/GLNS.jl`.

The solver takes several input parameters:

- A file describing the instance to solve. This file is written in the GTSPLIB format. It contains the cost matrix (with rounded coefficients, as the solver can only deal with integers), as well as the indexes of the different sets. Indeed, as we only want to reach each hole using exactly one configuration, we must specify which rows or columns of our matrix correspond to each hole. For example, if configurations q_1, q_2 and q_4 can reach hole h_1, the corresponding set from which to choose one single node will be $((q_1, h_1), (q_2, h_1), (q_4, h_1))$.
- The solver parameters. Three preset modes are available: $slow$, $default$ and $fast$. With the slowest mode giving better results at the price of computation time, the fastest being the opposite and the default mode being a balance between the two. As suggested by the research paper describing the solver, we will use the default settings unless we notice the computation times to be too high.

The output is a file containing the order in respect to which the nodes are visited. We shift this order so that it starts and ends with node 0 (the depot) and we build the final path by putting together the corresponding paths of the path matrix.

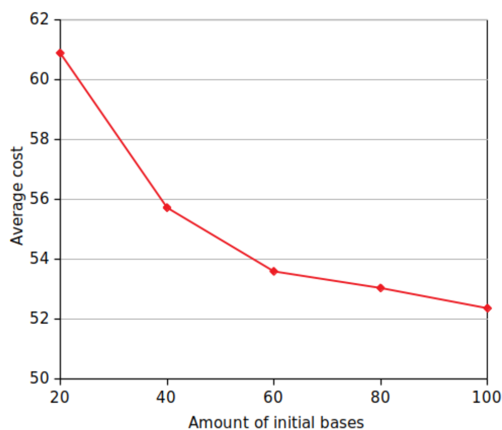
5.2.2.3 Results

As mentioned in section 3.4, the tests are initially done on only 20 holes out of the 240 ones of the part, in order to assess the performance of each method before trying it out on a bigger instance.

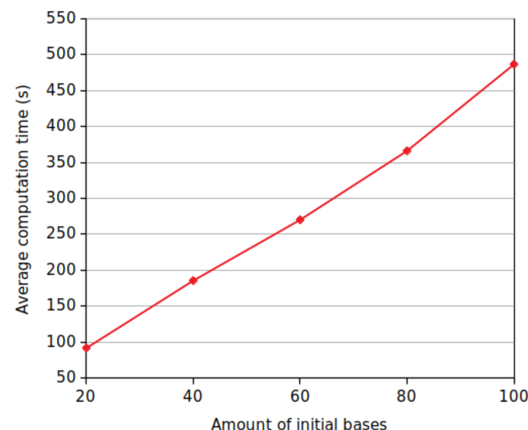
The general process for testing will be repeated as the internship goes on: we test each method ten times on ten different sets and compute average results. We prevent the influence of the randomness by using the same initial sets of configurations across different methods, in order to compare the average results.

For this first method, we can analyse the influence the amount of initial base poses has on the performance and the cost of our solution. To do so, we create 10 initial sets of configurations by generating one configuration per hole and we generate additional configurations on top of these. This process corresponds to incrementing N in algorithm 5. We go from $N = 1$ to $N = 5$ configurations per hole, which corresponds to a total amount of 20 to 100 base poses. The process is repeated 10 times and average results are computed.

Figure 5.4 below shows the corresponding results. While increasing the amount of configurations helps to lower the cost of the tour (figure 5.4a), this cannot be repeated infinitely. Indeed, while adding more poses helps to increase the amount of configurations to choose from, we notice that we quickly reach a point where the new configurations added will barely have an influence on the cost. However, increasing the amount of initial configurations will always have a negative impact on computation times (as seen in figure 5.4b). Indeed, going from 80 to 100 base poses only reduce the cost by about 1%, while the computation time increases by more than 30%.



(a) Evolution of the cost with the amount of initial base poses



(b) Evolution of the computation time with the amount of initial base poses

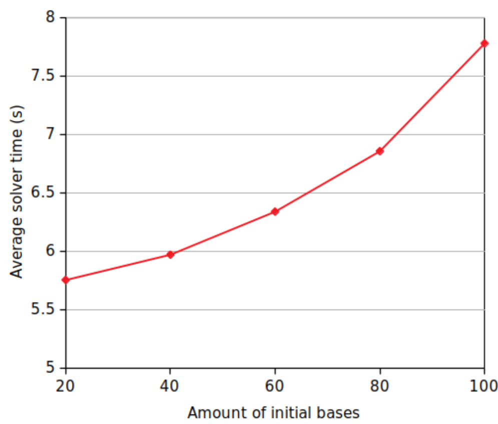
Figure 5.4: Average results of the method involving exact costs for different sizes of initial configuration sets

The total computation time shown in figure 5.4b includes:

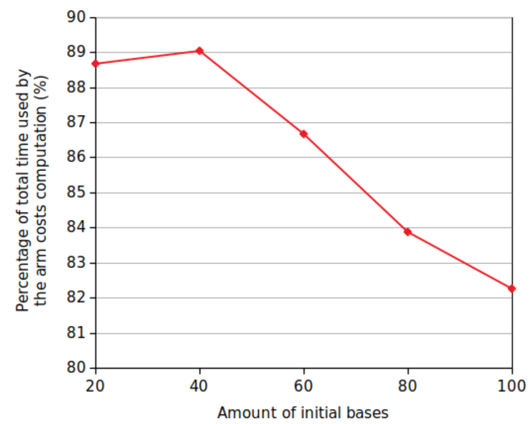
- The generation of the initial set

- The computation of the base poses cost matrix (and path matrix)
- The computation of the arm configurations cost matrices (and corresponding path matrices)
- The building of the two final cost and path matrices
- The generation of the file fed to the solver
- The time used by the solver
- The time used to associate the paths in order to build the final tour

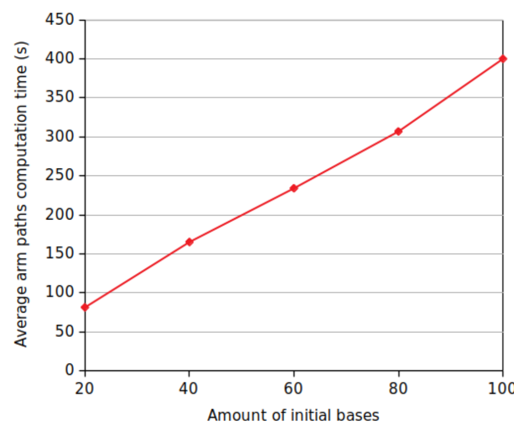
This total time is very high for such a small instance of 20 holes. Further analysis (figure 5.5) show that the high computation time is not due to the solver itself, with an average allocated time ranging from about 5.5s to 7.5s (figure 5.5a). Instead, more than 80% of the total time is used for the computation of the arm matrices (figure 5.5b), accounting for 80s to 400s of computation time (figure 5.5c).



(a) Evolution of the solver allocated time with the amount of initial base poses



(b) Evolution of the percentage of total time used by the evaluation of the arm costs with the amount of initial base poses



(c) Evolution of the time used by the evaluation of the arm costs with the amount of initial base poses

Figure 5.5: Further computation time analysis regarding the method involving exact costs

This high time is explained by the fact that we need to compute the paths associated to every single arm movement between the considered configurations. This is a very costly process which we need to optimise. Indeed, such high computation times on such small instances will increase drastically on bigger problems. We also notice that the part of computation time used by the path planning of the arms movements tends to lower. This is because other parts of the algorithm also tend to take longer with more nodes. Also, this time is not always guaranteed to increase similarly between two instances: the computation can sometimes be faster if a path is found early.

As for the performance of the GLNS solver itself, we notice a small increase of the computation time when adding more base poses. This is because the time represented in figure 5.5a is the total time used by the solver, which includes the solver start up time, the reading of the problem file, the actual solving of the problem and the writing of the solution. In reality, the time used to actually solve the problem is close to $1s$ and only slightly increases with the amount of initial base poses, which does not have a significant impact on the total time. Furthermore, it is important to mention that the solver is only a heuristic and that we do not have ways to compare its results to an exact GTSP solver.

The use of the $default$ solver setting is thus justified here because the computation times are kept low enough and are far from being the costliest part of this method, while the total amount of nodes ranges from about 100 to 1000.

5.2.3 Second approach with approximate costs

After analysing the computation times, we have to implement a way to drastically lower the time used to evaluate the costs (5.2.3.1). Results are shown in section 5.2.3.2.

5.2.3.1 Reducing the costs' evaluation time

We identified that the high computation times were due to the method used to evaluate the costs. Indeed, using the path planning algorithm developed by LAAS to evaluate all the costs associated to our problem is very costly, as it means several hundreds of paths need to be computed. As a result, we cannot use path computations to evaluate the cost associated to a movement between two nodes. Instead, we will try to approximate these costs in order to gain time and evaluate the impact it has on the results.

A simple way to approximate these costs is to consider the Euclidean distance between two configurations, instead of building a path and evaluating its length. This approach does not consider the collisions and corresponds to a linear interpolation between the two configurations, which happens to be a lower bound of any path between these configurations.

As a result, we will always underestimate the costs by using this approximation. Furthermore, since the arm configurations are usually close to each other, we expect the approximations to be a good representation of the real costs.

However, this approximation could not be applied to the movements between the base poses. Indeed, collisions matter a lot as these poses are scattered around the part. Approximating

the cost of their movement could lead to two base poses on either side of the part appearing as relatively close to each other while they are in reality separated by an obstacle (the part itself). Moreover, the time used to compute the movements between the base poses is not as time-consuming as for the arms, since the amount of base poses is much lower than the one of arm configurations. Also, the movements of the base are much simpler to compute, as it can only move forwards or backwards (in a straight line or while turning), whereas the arm has several joints which can move independently to each other.

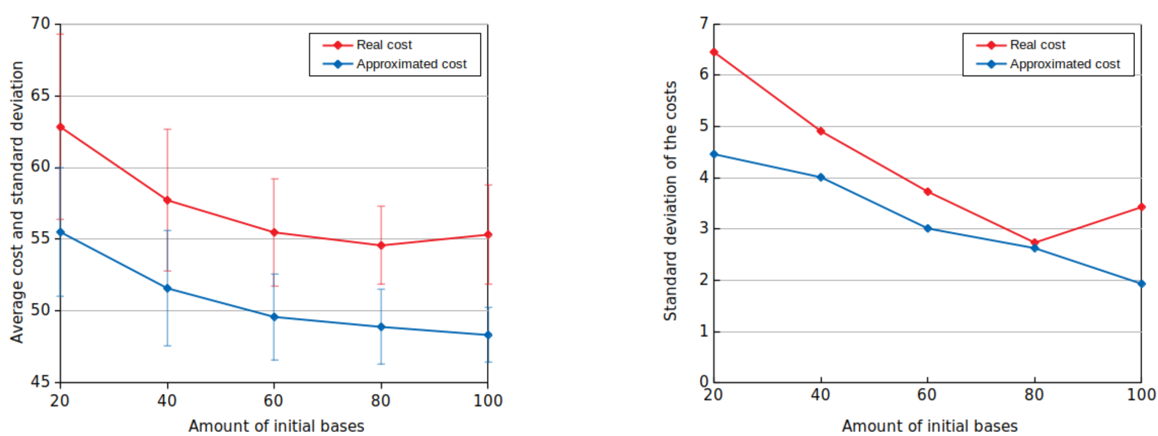
Consequently, we will only approximate the movements of the arms, by evaluating the Euclidean distance between two configurations instead of computing the paths. The new building process of the matrices is described in appendix C.3, and has a very similar structure to the algorithm which uses exact costs.

The solver will thus solve the problem using a partially approximated cost matrix. Once the tour is returned, the corresponding paths are either taken from the base paths matrix, or computed when movements between the arms are considered. In other words, this method only computes the paths of the arm movements included in the final tour, instead of all the possible paths, which is what was done in section 5.2.2.

5.2.3.2 Results

The tests for this method are done in the same way as in section 5.2.2.3 and from the same sets of initial data. The solver parameters remain unchanged, the only difference between the two methods being the way of evaluating the arm paths. The aim of this section is not to compare both methods but rather show the results of the approximate method separately. Section 5.2.5 compares all the methods.

Figure 5.6 shows the results regarding the optimality of this approximate method. Figure 5.6a shows how the cost of the tour evolves when increasing the amount of base poses.



(a) Evolution of the costs and their deviation with the amount of initial base poses

(b) Evolution of the standard deviation with the amount of initial base poses

Figure 5.6: Average results of the method involving approximate arm costs for different sizes of initial configuration sets

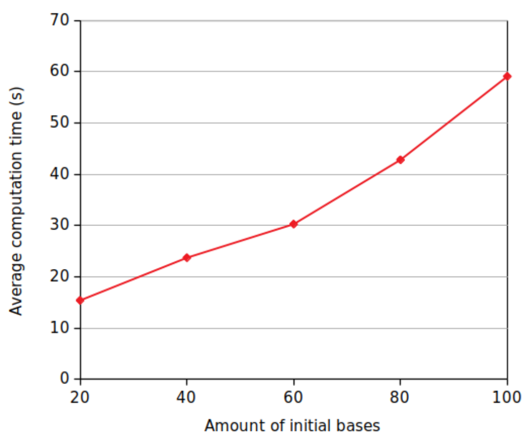
Two curves are shown on this graph:

- The approximated cost is the one returned by the solver, and is thus what we seek to minimize.
- The real cost, on the other hand, corresponds to the length of the real path associated to the approximate cost.

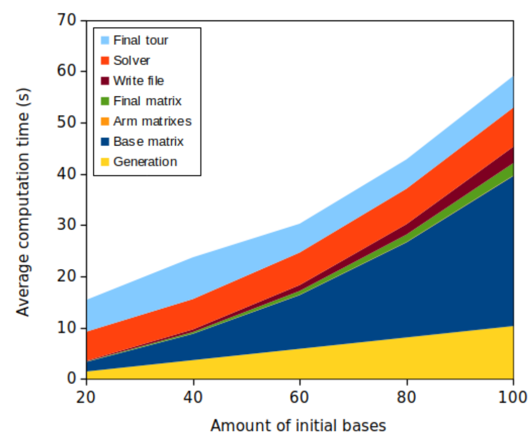
As mentioned earlier, the approximate cost is a lower bound of the real cost associated to the considered path, which is what we can see here. While there seems to be a constant difference between the two costs, it is important to remind that these curves are corresponding to average values. The difference is actually not constant. Indeed, the real cost tends to fluctuate more that its approximate counterpart, as shown by their respective standard deviation (figure 5.6b shows both deviations more clearly). This fluctuation is due to two reasons:

- A given approximated tour does not guarantee that the real movements will be as simple. For example, collisions can occur or the robot joints can reach their bounds. As a result, the length of two real paths can differ, even if their approximated length was initially the same, since this approximation cannot take the nature of the movement into account.
- Also, the path planning algorithm developed by LAAS does not always return the same path. Indeed, a test asking the algorithm to compute a path between the same configurations 10 times in a row led to a standard deviation of $\sigma = 2.5$ with an average real cost of 59.

Figure 5.7 below analyses the computation times regarding the approximate method. As seen in figure 5.7a, the computation time increases with the amount of base poses, but is far less concerning than it was with the previous method.



(a) Evolution of the computation time with the amount of initial base poses



(b) Evolution of the time-sharing between different parts of the algorithm with the amount of initial base poses

Figure 5.7: Average computation times of the method involving approximate arm costs for different sizes of initial configuration sets

Figure 5.7b shows a breakdown of the computation time shown in figure 5.7a. Each part of the full algorithm is shown. Chronologically:

- The time needed to generate the initial poses increases up to 10s for 100 base poses and will grow even more if we consider a part with more holes.
- Similarly, the time used to evaluate the costs associated to the base poses increases to the point where it becomes the biggest part of the total computation cost. We cannot however approximate these costs, as taking the collisions into account is necessary when we consider the base poses.
- Thanks to approximations, the time necessary to build the matrices related to the movements of the arms is barely visible on the graph.
- Similarly, the time spent to build the final matrix and the associated file are not significant for a problem of this size.
- The time needed by the solver is almost constant, as shown in figure 5.5a.
- Finally, the time necessary to compute the chosen arm movements and associate them with the already computed base movements does no depend on the amount of initial base poses, but rather on the amount of holes. Indeed, the more holes we consider, the more paths will be needed to compute. The initial bump seen on the graph is due to the fact that the path planning algorithm sometimes takes longer to connect 2 given configurations.

With good computation time results, we try to improve the cost of the tour given by this method in section 5.2.4 below.

5.2.4 Third approach: approximate costs and iterations

As described previously, one of the major drawbacks of the method involving approximate costs is the fact that it does not give us the best possible tour. We try to improve the results by iterating this method (5.2.4.1), results are shown in 5.2.4.2

5.2.4.1 The iterative process

The goal of this method is to improve the results given by the approximate method (5.2.3). In order to do this, we first run the approximate method once, which means generating the base poses, computing and assembling the arms and base matrices together in a file, solving the problem and building the final tour. Then, we try to explore other possibilities by visiting other tours with an incrementing approximate cost, in order to see if the corresponding real tour is improving. Algorithm 2 highlights this process.

First, we get the approximate tour, the real tour and the cost matrix from a run of the approximate method (lines 1 - 3) on our problem instance. Then, we update the approximate coefficients of the cost matrix with their real values from the built tour (line 4). Lines 5 - 11 corresponds to the iterations. In each iteration we solve the problem using the updated matrix and compute the associated tour (lines 6 - 7) . If this tour is better than the one previously

Algorithm 2: Iterating over the approximate method

Input: $problem, max_iter$
Output: $best_tour$
1 **Initialize:** $nb_iter \leftarrow 0$
2 $costs_matrix^{approx}, approx_tour, real_approx \leftarrow ApproximateMethod(problem)$
3 $best_tour \leftarrow real_approx$
4 $UpdateMatrix(costs_matrix^{approx}, real_tour)$
5 **while** $approx_tour < best_tour$ *and* $nb_iter < max_iter$ **do**
6 $\quad approx_tour \leftarrow SolveGTSP(costs_matrix^{approx})$
7 $\quad real_tour \leftarrow BuildPath(approx_tour)$
8 $\quad$ **if** $real_tour < best_tour$ **then**
9 $\quad\quad best_tour \leftarrow real_tour$
10 $\quad UpdateMatrix(costs_matrix^{approx}, real_tour)$
11 $\quad nb_iter \leftarrow nb_iter + 1$
12 **return** $best_tour$

found, we update the best tour (lines 8 - 9). Either way, the cost matrix is updated (line 10). As for the stopping criteria (line 5), we either stop when the current approximate tour gets worse than the best real tour found, or after a given amount of iterations. The first criterion can be explained because the approximate tour is always shorter than its real counterpart. As a result, the GTSP solver will find an approximate tour shorter than the best one, until a point where the costs worth looking at are all updated and we fall back on the best tour again.

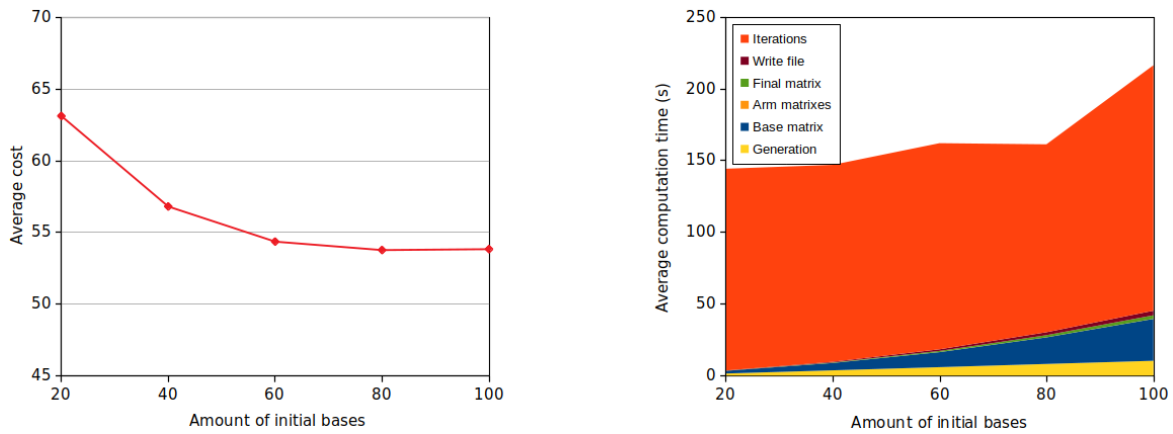
Note: We only build the paths that have not been computed yet because it is unnecessary and time-consuming to build the path between the same nodes twice.

With the iterative method defined, we can test it and analyse the results.

5.2.4.2 Results

The testing conditions for this method are the same as mentioned in 5.2.2.3. We choose an amount of 10 maximum iterations here, the goal being to explore the neighbourhood of the initial tour and not the whole matrix.

The results are shown in figure 5.8. Figure 5.8a shows the evolution of the cost of the tour, which evolves similarly to the two other methods. As for the computation times (figure 5.8b), we can see how the iterations time adds itself on top of the time used by the initial run of the approximate method (the initial solving and building times are included in the iterations here). We previously noticed (figure 5.7b) that $10s$ to $15s$ are needed to write the file, solve and compute the tour, which would translate to $100s$ to $150s$ if we iterate this process 10 times. This closely corresponds to the time increase we can see on figure 5.8b.



(a) Evolution of the cost with the amount of initial base poses

(b) Evolution of the time-sharing between different parts of the algorithm with the amount of initial base poses

Figure 5.8: Average computation times of the method involving iterations over approximate arm costs for different sizes of initial configuration sets

This method has major drawbacks related to computation times, as each iteration is costly time-wise and will be even more so on bigger problems. Indeed, both the solving and tour building times will increase on problems with more holes, as the solving time depends on the amount of nodes, and the tour building time directly depends on the number of holes. The performance of this method is compared with the two other ones in section 5.2.5 below.

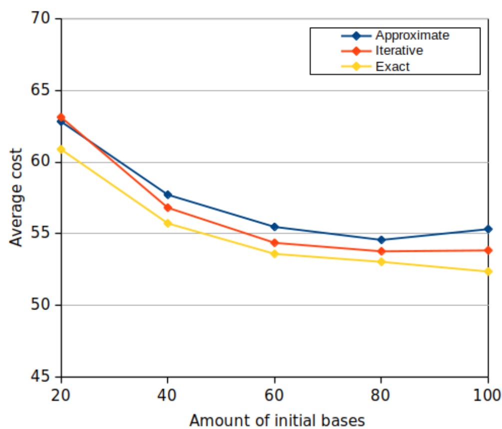
5.2.5 Comparative results

Now that the three methods starting from the generation of an initial set of base poses have been described and tested, we can compare the corresponding results. The tests have all been done from the same sets of initial poses. The results are shown in figure 5.9 below.

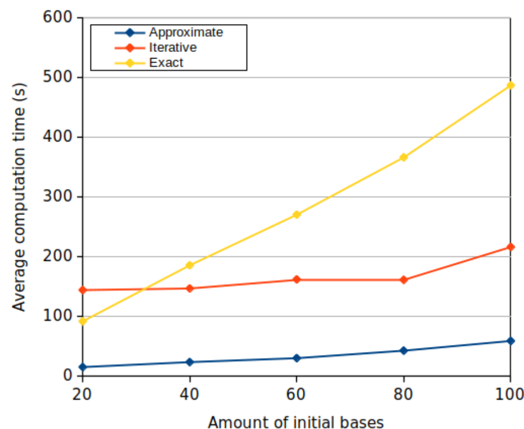
Figure 5.9a shows that the method involving exact costs gives the best results regarding the tour, while the approximate method gives the worst ones and the iterative method helps to reduce the cost of the former. In theory, the iterative method could reach the same cost as the exact method with the max iterations criterion removed.

However, figure 5.9b shows that the approximate method is by far the most interesting one regarding computation times, being about 10 times faster than the exact one while having only about 5% of loss on the costs. As for the iterative method, we can clearly see the increase due to the iterations over the approximate method. It is however important to mention that while this increase may look constant over this set of data, it will not be on bigger problems. Indeed, we saw that the increasing amount of base poses did not have a noticeable influence on the solver time on such a small scale problem of 20 holes, but they will on bigger instances.

From these results we can already exclude the exact method as its computation time will sky-rocket if we consider the 240 holes of the studied part. Indeed, many more paths would have



(a) Evolution of the cost with the amount of initial base poses



(b) Evolution of the computation time with the amount of initial base poses

Figure 5.9: Comparison of average results of the 3 methods for different sizes of initial configuration sets

to be computed. While the iterative method could be an interesting variant of the approximate one, its computation time will also greatly increase on bigger instances, as every iteration will be very costly.

As a result, we will only test the approximate method on the whole part before considering comparing it with the iterative one. This testing is shown in section 5.2.6.

We also analysed the actual tours in Gepetto Gui. With the 20 holes being split up in two subsets of 10 holes each, on either sides of the part, the minimum amount of base poses needed to deburr all the holes was 2. This is the amount we noticed in almost all the tests, excluding some cases where generating 1 base pose per hole was simply not enough to get a good covering of the holes. This validates the choice of $K_p = 10$ for the additive weight on the movement of the base poses, since the minimum amount of base poses was found every time.

As for the movements of the arms, we can see that they are far from being optimal, as the movements between two consecutive holes are usually more complicated than what we could picture ourselves, even with the use of the exact method. This means that there is not enough choice regarding the arm configurations. Indeed, while the solver chooses the best movements from the initial set, this set is simply not dense enough to provide smooth movements of the arms. Section 5.3 deals with this issue by exploring ways to improve the movements related to the arms.

5.2.6 Tackling bigger scale problems

In this section, we test the approximate method described in 5.2.3 on the entire part with 240 holes in order to see how the computation times evolve. Figure 5.10 shows a breakdown of the total computation time, from the initial generation to the building of the final tour.

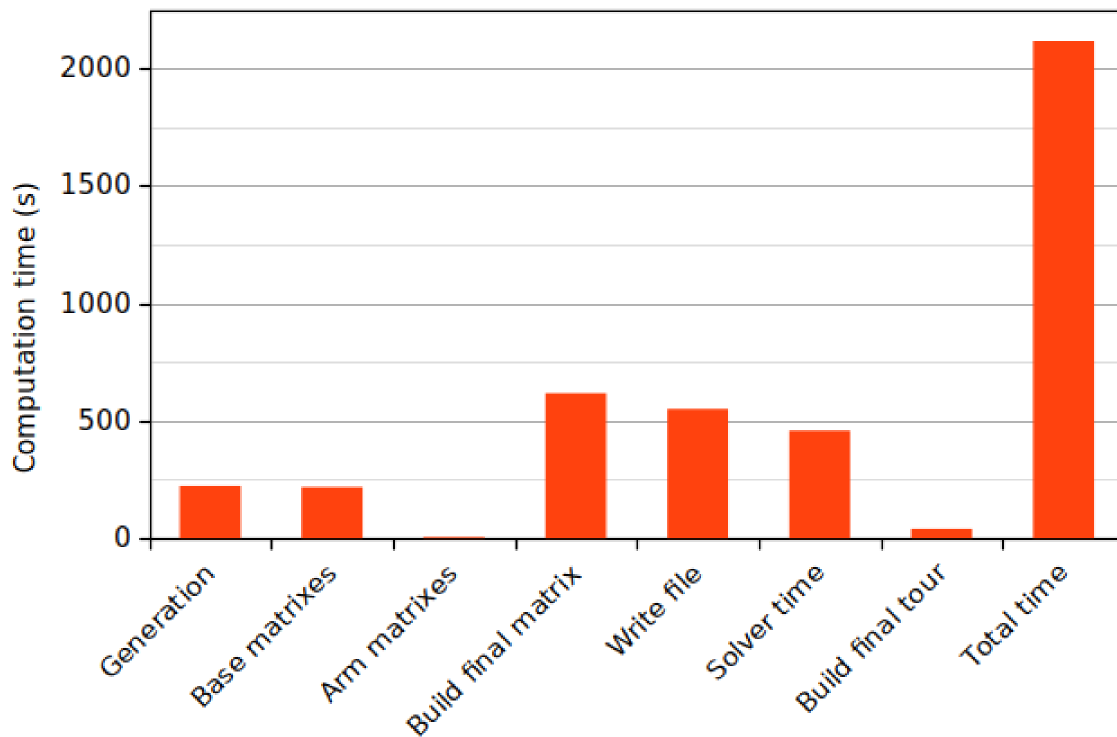


Figure 5.10: Time computation breakdown on a 240 holes part - approximate method

We can see how each part of the algorithm now behaves with 240 nodes and around $12,000$ nodes.

- The time needed to generate the initial poses is quite high with around $250s$. While the time needed to generate N configurations per hole ($N = 1$ here) is really low (less than $5s$), the part where we check if each base pose can reach each of the other holes is by far what takes the most amount of time in this algorithm. If we want to keep using this algorithm, we will have to find ways to reduce its computation time.
- The time used to compute the base poses is also high. Yet, as mentioned previously, we cannot approximate these movements.
- Thanks to approximations, the time necessary to build the matrices related to the movements of the arms is kept very low.
- The time spent to build the final matrix and the associated file are using a significant amount of computation time on such a big instance. This is explained because both algorithms have to deal with matrices of size about $12,000$.

- The time needed by the solver is also significant, with around 100s needed to simply read the input file, the rest of the time being majorly used to actually solve the problem. The $default$ solver setting has a 360s timeout which is used in order not to spend unnecessary time trying to slightly improve the cost. The timeout has been reached here.
- Finally, the time necessary to compute the chosen arm movements and associate them with the already computed base movements took about 40s and cannot be improved, as it is necessary to connect the 240 holes and the depot at some point.

This graph shows that the approximate method, although much faster than the other 2 mentioned, is not really adapted to a problem of this size. Furthermore, there isn't a lot of room for improvement regarding its computation times. This also proves that the iterative method isn't worth testing, as one iteration takes about 500s if we do not consider the time necessary to write the file, which could theorically be reduced a lot if we only change a few values inside it.

As a summary, only the approximate method is worth using on bigger scale problems, but it still has high computation times. It also does not give the best results, which can fluctuate a lot depending on how the tour is built. Also, while we mentioned that there often was not enough arm configurations to choose from, adding more nodes to an already big problem does not appear to be the safest approach. In other words, we have to find ways of improving the movements, while using much less nodes.

5.3 Starting from an initial set of arm configurations

This section describes our second approach which corresponds to feeding the GTSP with a large initial set of arm configurations. A more accurate breakdown of this second part of the internship is described at the end of the introduction below (5.3.1).

5.3.1 Introduction

After noticing quite poor visual results regarding the arm movements, we decided to see if increasing the amount of arm configurations could lower the final cost. Instead of unnecessarily adding more nodes, we decided to start from an already computed tour, retrieve the chosen base poses used in the final tour, as well as the associated arm configurations. From there, we tried to generate more arm configurations so that each base pose could reach each of their associated holes in multiple ways. We ran the solver again from this instance to see whether the cost was improved. We tested this on one already computed instance of the series of tests with 20 holes and 80 generated configurations, where we added 3 arm configurations for each hole on top of the existing one. Table 5.1 below shows the results.

Method	Cost	Arms cost	Computation time	Nodes
Initial approximate method	56.95	20.95	44.00s	734
Extension with more arm configurations	50.13	15.16	12.05s	81

Table 5.1: Comparison between the initial approximate method and its extended variant

The results show an improvement of the cost. Indeed, thanks to the generation of additional arm configurations, the cost related to the arms was improved by around 27% and the total cost by about 12%. Additionally, since only 2 base poses were chosen among the initial 80, the amount of nodes was drastically reduced, leading to much faster computation times. The amount of nodes in the initial methods is:

$$n_{nodes} = 1 + \sum_{q_i \in Q, i \neq 0} n_{holes|q_i}$$

Where Q is the set of base poses and $n_{holes|q_i}$ is the number of holes reachable by base pose q_i. The depot q_0 cannot reach any holes but counts as one node. Due to the distribution of the considered holes (10 on either side of the part), we can suppose that each base pose can reach 10 holes. Therefore, with the initial method and 80 base poses considered and one arm configuration per hole:

$$n_{nodes} = 1 + \sum_{i=1}^{80} 10$$
$$n_{nodes} = 801$$

We find out more nodes than in the example shown above, this is because not all the base poses could reach 10 holes. As for the extension with more arm configurations, 2 base poses were chosen, each of which could reach a given hole 4 times (as 3 arm configurations were added). The amount of nodes is then:

$$n_{nodes} = 1 + \sum_{i=1}^{2} 10 \times 4$$
$$n_{nodes} = 81$$

Note: the reason why the difference between the costs are not the same ($56.95 - 20.95 \neq 50.13 - 15.16$) is because the base matrix was computed again during the extension. As pointed our earlier, the path planning algorithms do not always find the same paths, which explains this difference.

This first test shows that focusing on the movements of the arms by adding more arm configurations can both reduce the cost related to the arms and the computation times and therefore motivates the use of this method. It is however necessary to choose the initial base poses. Here we used an already existing tour to get these base poses, but we saw that the use of the approximate method applied on a large amount of base poses was not effective time-wise. We need to come up with new efficient ways of selecting the base poses, at the risk of ending up with an overall higher cost than the approximate method applied on the initial set of base poses.

We thus implement and test several methods of base poses selection, on top of which we run the extension involving more arm configurations described above, in order to compare them together and with the approximate method previously implemented. The general approach is described in section 5.3.2. The base poses can either be selected from a clustering approach (5.3.3) or from the generation algorithm we previously implemented (5.3.4). Comparative results between all the methods implemented are shown in section 5.3.5. Finally, section 5.3.6 compares our algorithms with the approach described in [4].

5.3.2 General principle

This section presents the general principle of the approach involving the generation of additional arm configurations. This principle is summarized in figure 5.11.

- First, we select the N base poses to be used in the final tour, as well as the arm configurations linking each of the base poses with each of the holes it is able to reach. This step is not described here as it is the point of the upcoming sections.
- From there, two cases appear depending on whether we have a partition of the holes or not.

A partition of the holes means that each hole is included in exactly one cluster. In other words, each hole can be reached by one single base. If such a partition exists, this means that we can actually solve N separate GTSPs. Indeed, with each hole being allocated to one single base, we can split up our problem in several smaller ones, which will lead to faster computation times. As for the base poses, we can solve a TSP in order to find the fastest tour among them (including the depot). Moreover, if such a partition does not exist, we can either find a way to make one by allocating the holes appearing in several clusters to only one of them, and then solve the problem using the method described above. If we decide not to make a partition, we will have no choice but to solve one single GTSP.

No matter the case, we generate K additional arm configurations for each hole reachable by a given base pose, in order to have more configurations to choose from. Indeed, we add K more ways for each base pose to reach each of its associated holes, to the already known arm configuration. We suppose that we already know which holes can be reached by a given base pose, as well as the corresponding arm configuration, as it is a much-needed criterion to actually select the base poses. The generation algorithm uses the same functions and methods as algorithm 5, where for each base pose and each hole it can reach, we try to generate K additional pre grasp and grasp configurations to the already known arm configuration. As a result, each base pose can deburr each of its reachable holes in $K + 1$ different ways. Also, the final tour is built at the end in both cases.

Whether we have a partition depends entirely on the method used to select the base poses. For example, if we select them from an already existing tour, we will indeed have a partition of the holes, as each hole only gets visited once in the final tour.

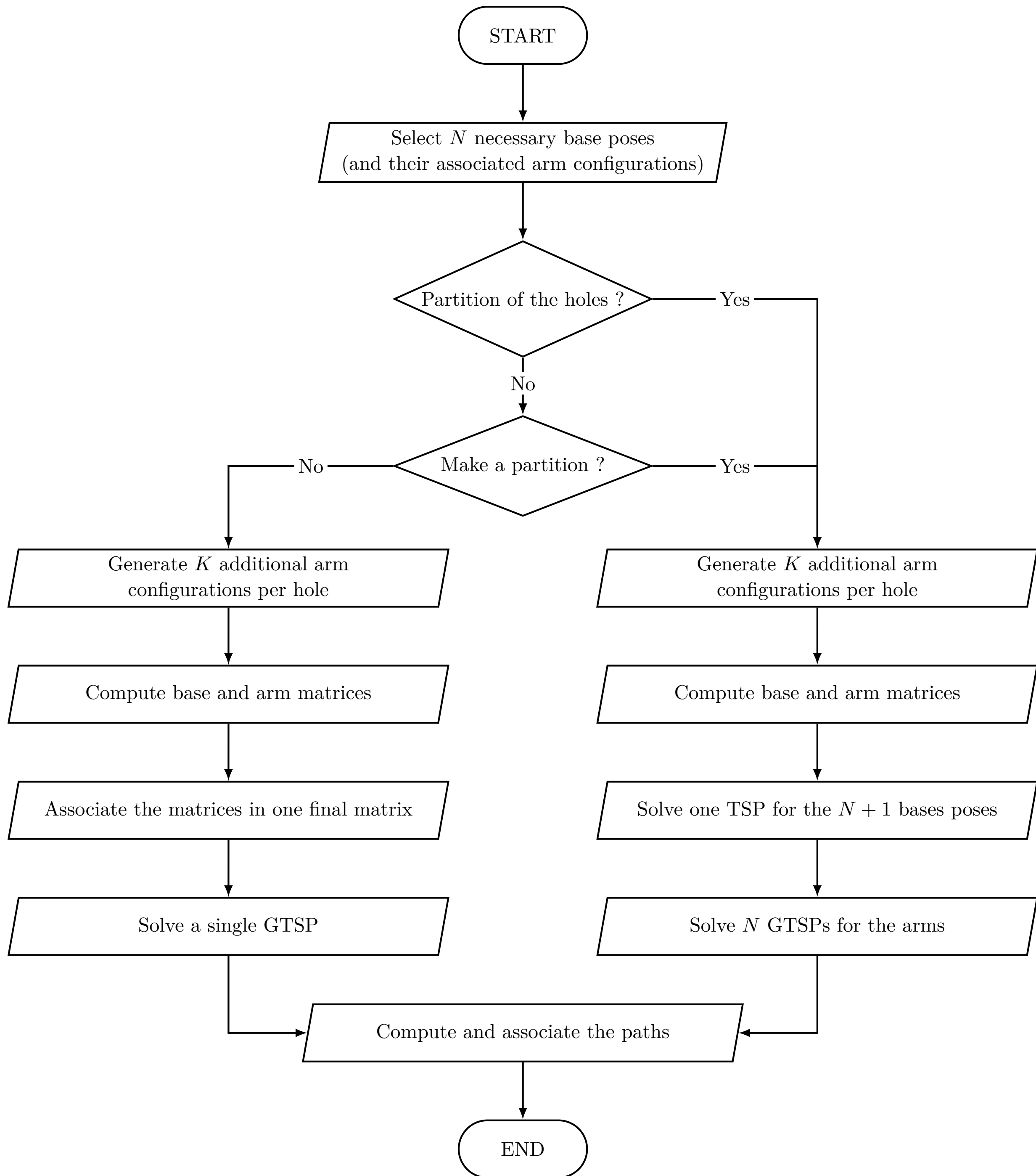


Figure 5.11: General principle of the approach involving the generation of additional arm configurations

5.3.3 Clustering using LAAS algorithm

The first method tried uses LAAS's already existing clustering algorithm. This algorithm directly gives N base poses which can cover all the holes. The output corresponds to a partition of the holes, which is a good point. The method involves computing a base pose for a given hole and trying to reach as many holes as possible from there; the process is then repeated until all the holes are covered. Algorithm 3 below outlines the process.

Algorithm 3: LAAS algorithm to find clusters

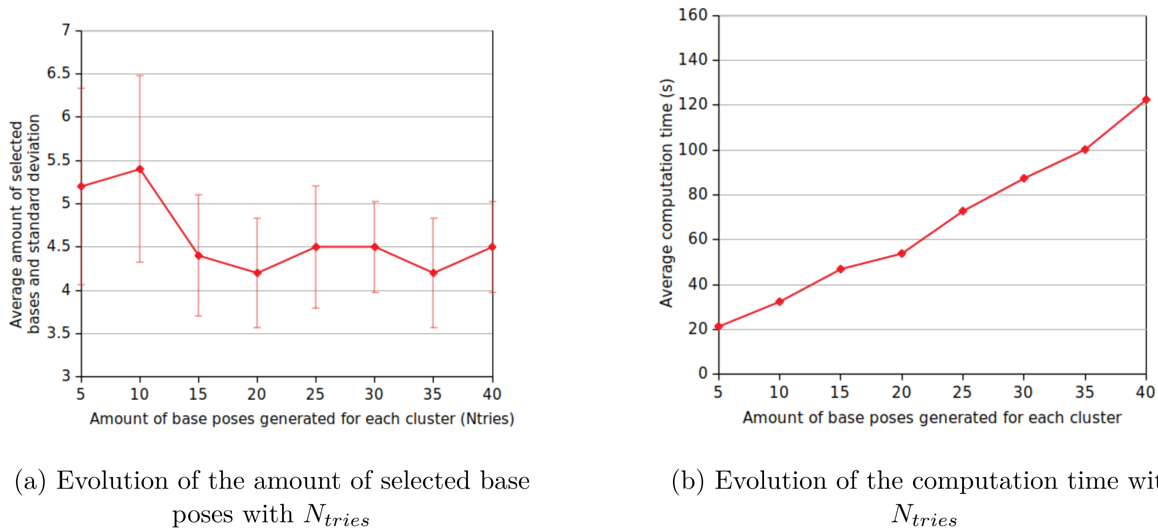
```
  Input: N_tries, holes
  Output: clusters
1 Initialize: clusters ← ∅, remaining_holes ← copy(holes)
2 while remaining_holes do
3     h_start = choice(remaining_holes)
4     cluster = find_cluster(h_start, remaining_holes, N_tries)
5     add cluster in clusters
6     foreach h_i, qp_i, q_i ∈ cluster do
7         remove h_i from remaining_holes
8 return clusters
```

The general principle is quite simple: a random hole is chosen among the list of holes (line 3), a cluster is then found around this hole (line 4). The $find_cluster$ function aims at finding the biggest cluster starting from the chosen hole. This is done by generating a configuration reaching this hole and checking how many other holes the associated base pose can reach. The process is repeated with N_{tries} different base poses and the one able to reach the most holes is returned. The contents of the $find_cluster$ algorithm are described in appendix C.4. A cluster contains the (h_i, qp_i, q_i) sets, where each set contains the pre grasp (qp_i) and grasp (q_i) configurations associated to hole h_i. All sets within the same cluster correspond to a unique base pose, which can reach all the holes of the cluster. From there, the remaining holes (i.e. the ones who do not have a base pose associated to them yet) are updated (lines 6 - 7). The process is repeated until all the holes are covered (line 2).

We can test the influence the value of N_{tries} has on the chosen base poses and the computation time. While we will compare all the base selection methods later (5.3.5), the goal of this test is to study the impact of N_{tries} in order to properly tune the method when doing the comparative testing. The aim is therefore to find which values lead to the least amount of base poses, with the fastest computation times. As randomness has a major impact on the outcome of this method (both regarding the generation of configurations and the choice for the starting holes), we repeat each test involving a given parameter value 10 times. We consider the 240 holes for the testing, since the difference between the computation times will be more noticeable.

Figure 5.12 below shows the influence N_{tries} has on the amount of selected base poses, and the corresponding computation time. Figure 5.12a shows that increasing N_{tries} (i.e. increasing the amount of potential clusters among which the best one is selected) has an influence on the amount of selected base poses up to a point where both entities become uncorrelated. Indeed, the average amount of selected base poses lowers when N_{tries} increases, up to $N_{tries} \approx 20$ where it starts to even out. This means that $N_{tries} = 20$ is enough to get the least amount of base

poses. While one could think that increasing its value even more could potentially lead to more robust results, it is important to note that the standard deviation appears to be roughly the same for $15 < N_{tries} < 40$, which means increasing N_{tries} does not improve the robustness. Moreover, increasing N_{tries} also lead to a noticeable increase in the computation times. Because of these reasons, we will set N_{tries} to 20 for the rest of this study.



(a) Evolution of the amount of selected base poses with N_{tries}

(b) Evolution of the computation time with N_{tries}

Figure 5.12: Influence of the value of parameter N_{tries} in the clustering algorithm

While this process uses similar functions as the ones we described in section 5.2.1, it uses a heuristic to create the clusters, as the clustering decisions are never questioned. As a comparison, the generation algorithm we initially implemented (section 5.2.1) computes and saves one (or more) cluster(s) for every hole of the part from which we could then find the best ones. As the clustering heuristic described here seems to oscillate between 4 and 5 chosen base poses, we aim at getting more robust results by re-using our initial generation algorithm. Section 5.3.4 below explores ways of selecting the least amount of base poses ensuring a full coverage of the holes, by combining the generation algorithm with other methods.

5.3.4 Selecting the base poses found by the generation algorithm

After testing the clustering algorithm, we noticed some fluctuations regarding the amount of chosen base poses. As our goal is to try to select the least amount of base poses, we seek to improve the results shown above by using our initial generation algorithm and associating it with different methods. In section 5.3.4.1 we try to make the algorithm less time-consuming, before coupling it with a covering algorithm (5.3.4.2). Finally, section 5.3.4.3 explores another approach aimed at reducing the generation time even more, using a K-medoids clustering approach.

5.3.4.1 Making the generation algorithm more efficient time-wise

As seen in section 5.2.6, it takes about $250s$ for the generation algorithm to deal with a 240 holes part. More specifically, this computation time includes:

- The generation of 1 configuration for each of the 240 holes.
- Checking whether each of the associated base poses can reach all the other holes of the part.

Further analysis show that the initial generation takes less than $5s$ for such a problem. The second part is therefore by far the most time-consuming part of the algorithm. This is obviously explained by the massive amount of configurations to explore, as we check if each of the 240 base poses can reach each of the 240 holes. When testing if a base pose q can reach a given hole h, the algorithm developed by LAAS works as follows: N random configurations are tried to be generated (from base pose q) without taking the collisions into account. Then, the first configuration satisfying all the constraints is returned. Thus, in the case where h cannot be reached, the algorithm will spend a lot of time to try an compute the N configurations and will eventually reach its maximum amount of allowed iterations. In other words, an unreachable hole is very costly, as the generation algorithm will spend a lot of time computing worthless configurations.

As our study is aimed at solving problems where several base poses are needed to reach all the holes, a lot of holes are therefore not reachable by a given base, which leads to high computation times. We implement a way to not consider all the holes, but only the closest ones to a given base pose.

In order to do this, we can pre-calculate the error associated to the constraint between an existing base pose and a hole. Indeed, given that we know base pose q_1 can reach hole h_1 in the pre-grasp configuration qp_1, we can evaluate the error associated with reaching all the other holes, from this existing configuration qp_1, without having to spend time generating additional configurations. As the first part of the generation algorithm associates at least one base pose to each of the holes, we can easily access the error associated to the other holes. This error takes into account the distance and the angle between configuration qp_1 and a given hole of the part. We can iterate the process in order to obtain the error values associating qp_1 to all the holes of the part (excluding h_1 since we already know qp_1 can reach h_1 thanks to the initial generation).

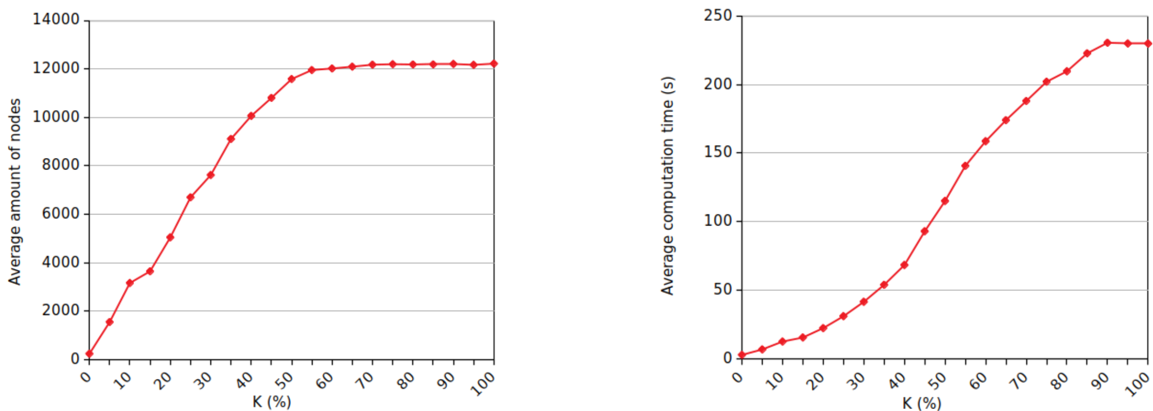
We can then repeat the process for all the generated configurations and store the results in a matrix. Coefficient $(i, j), i \neq j$ of the matrix tells us about the error linked to reaching hole h_j from the base pose generated for h_i. The matrix is asymmetrical, as there is no reason qp_i associated to h_i can reach h_j as easily as qp_j associated to h_j could reach h_i. *Note:* if several configurations were initially generated for each hole, we only consider the first one.

We now have a matrix containing all the errors. We can now define a threshold value t_{hr} so that we will only try to reach the holes with an error lower than this value. We denote e_m and e_M respectively the minimal and maximal values of the error matrix (without considering the diagonal coefficients). We can define the threshold by:

$$t_{hr} = e_m + (e_M - e_m) \times K \tag{5.9}$$

Where K helps to tune the threshold value. $K = 1$ will consider all the holes for example, as it leads to $t_{hr} = e_M$. We can now test the influence of K on the computation times and on the amount of nodes. It is in fact too early to consider the amount of selected base poses, as such a method has not been implemented yet. Instead, we will try to tune K in order to get a relatively close amount of nodes as if we used $K = 1$, which should allow to reduce the computation times without having too much of an influence on the base selection.

Figure 5.13 below shows the average results. From a single starting set of base poses, we tested the influence of different values of K, where each test was repeated 5 times. As the generation algorithms try to compute random configurations to reach the holes, it is necessary to repeat the process multiple times to get suitable results. This was only tested on a single initial set of base poses, since what matters the most is the position of the holes and not the base poses. We can see (figure 5.13a) that values of K above 55% lead to an amount of nodes very close to the one found with $k = 100\%$ (around $12,000$ nodes). Moreover, figure 5.13b shows an ever-increasing computation time, which means that we have to choose the lowest possible value for K, if we want to have the best computation times. However, while losing around 10% of the nodes could seem to be an acceptable trade-off, it is actually not a good idea. Indeed, these nodes correspond, for each base pose, to the farthest holes it can reach. Ignoring these nodes could lead to a higher amount of base poses needed to cover all the holes. For this reason, we will choose the value $K = 55\%$ fo the rest of our study.



(a) Evolution of the amount of nodes with K

(b) Evolution of the computation time with K

Figure 5.13: Influence of the value of parameter K in the generation algorithm

The chosen value of K therefore helps to reduce the computation times by around 40%. This value of K can depend a lot on the problem we deal with. Indeed, if we want to deal with deburring holes that can all be reached by a single base pose, setting K to any value below 1 will have an impact on the cost, as holes that could be reached will not be visited, and the amount of chosen base poses will therefore increase. *Note:* while the improvement of the generation algorithm was not included in the tests shown in section 5.2, it is important to mention that

these methods would still be time-consuming due to the high amount of nodes and its impact on other parts of the algorithms.

5.3.4.2 Covering algorithm applied on the initial base poses sets

With the computation times regarding the generation algorithm being drastically reduced, we can now try to select the base poses we will use to build our final tour. While we know that solving the GTSP directly from this set of base poses will lead to a tour from which we can select the base poses, we saw that this process is very costly time-wise.

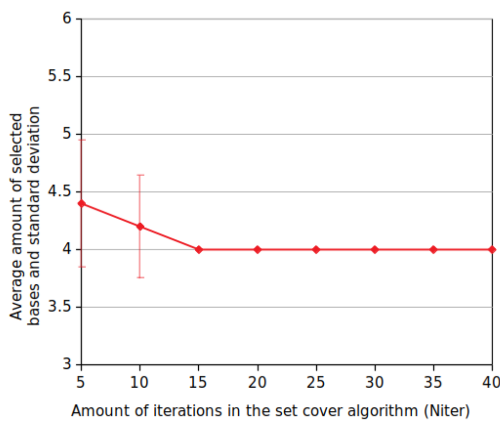
At this point we know which holes can be reached by each of the base poses generated. Our goal from there is to select the least amount of base poses that can reach all the holes. This can be seen as a Set Cover Problem (SCP). Indeed, we have different subsets of the holes (each corresponding to a base pose), and we want to find the smallest union of these subsets that can cover all the holes. Although the SCP is a $\mathcal{NP}$-complete problem, a well known greedy algorithm to solve it exists [7]. It iterates by choosing the biggest subset at each stage. While this method offers fast computation times, it is also far from being optimal. A few tests were ran using this method and showed quite poor results, as expected, with the chosen sets being mostly of size 5 or 6, for a problem that can be covered with 4 base poses.

We then tried to look for other algorithms used to solve these problems and found a study allying the greedy algorithm with Lagrangian Relaxation approximations. The algorithm described in [8] claims to be 99% optimal, with no mention of time complexity and is available through a Python module. The solver iterates its combined greedy and Lagrangian Relaxation method until a maximum amount of iterations is reached.

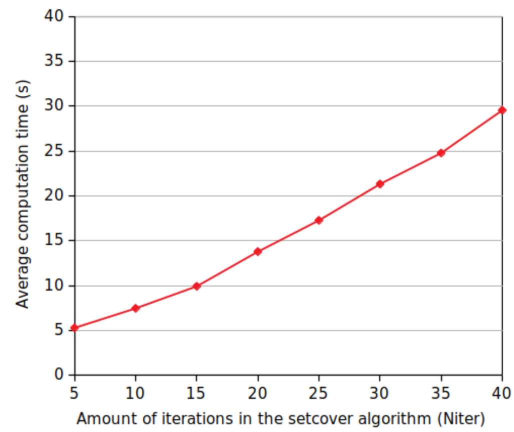
We therefore study the influence of this maximum iterations parameter (which we will denote N_{iter}) on both the computation times and the amount of selected base poses. In order to do this, we start from an already generated set of 240 base poses, on top of which we run the set cover algorithm 5 times for different values of N_{iter}, ranging from 5 to 50 iterations. While SCPs also usually associate a cost to each set, we cannot define a proper one in our case. Indeed, while we can evaluate the cost linked to moving between 2 different base poses, we cannot associate a similar cost to a single base. While we could consider adding an average value of the costs related to the approximated arm movements needed to reach all the holes of the set, this idea is not suitable, as we aim to further generate additional arm poses in order to reduce the said cost. As a result, each set will be associated with the same cost.

Figure 5.14 shows the results. We can see of figure 5.14a that the average amount of base poses starts to be equal to 4 as soon as N_{iter} is high enough. The associated standard deviation is in that case equal to 0, which means all the results returned for $N_{iter} > 15$ had the same amount of base poses. This corresponds to the announced 99% optimal results, as long as N_{iter} is high enough. While setting N_{iter} to 10 could seem like a good compromise, it is important to mention that we do not want to risk ending up with an amount of base poses different of the optimal covering value. The value of parameter N_{iter} will therefore be set to 15. As for the computation times, the algorithm can solve such a big instance of 240 sets in about $10s$ (figure 5.14b).

One of the main issues regarding this method is the fact that no base selection criterion is used. Indeed, while the algorithm returns a good amount of subsets, we do not know if another union



(a) Evolution of the amount of base poses with N_{iter}



(b) Evolution of the computation time with N_{iter}

Figure 5.14: Influence of the value of parameter N_{iter} in the Set Cover algorithm

of same size exists. This other union could quite possibly lead to a better cost of our final tour. Indeed, there is no way to select the best set of base poses using the Set Cover algorithm, as we could not associate a cost to each set. We therefore build an iterative process from which we will select several covering sets using the algorithm. From there, our goal is to find the best one.

As the cost related to the base poses will not be improved when solving the final(s) GTSP(s) but the one related to the arms will be, our aim is to select the subset of base poses leading to the smallest distance without considering the arms. We can describe this problem as a GTSP where we set the cost of the arm movements to 0. This means that the solver will look for the tour among the base poses while trying to cover all the holes, regardless of the cost needed to actually deburr the holes. Although this problem has quite a high amount of nodes, the computation times are very fast as almost all the costs are the same. We will use the $fast$ mode of the solver, as the aim is to quickly find a fast tour, without having to look for small improvements which are indeed not doable in this case. Algorithm 4 below highlights the process.

Algorithm 4: Base selection using multiple Set Cover algorithms and a GTSP

Input: $problem, N_{runs}, N_{iter}$
Output: $best_sub_problem$

1. **Initialize:** $sub_problem \leftarrow \emptyset, best_sub_problem \leftarrow \emptyset$
2. **foreach** $i \leftarrow 1$ *to* N_{runs} **do**
3. $bases \leftarrow SetCover(problem, N_{iter})$
4.   remove $bases$ from $problem$
5.   add $bases$ to $sub_problem$
6. $cost_matrix \leftarrow BuildCostMatrix(sub_problem)$
7. $tour \leftarrow SolveGTSP(cost_matrix, mode = fast)$
8. $best_sub_problem \leftarrow GetBasesFromTour(tour)$
9. **return** $best_sub_problem$

Given an initial $problem$, which contains the data related to the base poses and the holes they can reach, we first extract a $sub_problem$ from it by running multiple Set Cover algorithms

(lines 2 - 5). We obviously remove the chosen base poses from the initial *problem* at each stage, in order not to select the same base poses twice. We now have a *problem* containing the N_{runs} most interesting sets of base poses, from which we select the best ones by running a GTSP (lines 6 - 8) which does not take the arms movements into account (not represented here). The influence of N_{runs} will be discussed in section 5.3.5.

The output of both methods we described using the Set Cover algorithm do not correspond to a partition of the holes. Indeed, as the generation algorithm always considers all the holes reachable within the threshold (see equation 5.9), two different base poses can reach the same hole. After adding more arm configurations, if we follow the flowchart reminded in figure 5.11, we can either solve the problem with a single GTSP, or turn it into a partition in order to divide it in multiple sub-problems.

We will call "conflicting holes" the ones which can be reached by more than one base pose. If we want to build a partition, we must allocate the conflicting holes to the most suitable one. In order to do this, we can build a distance matrix between all the holes. Then, from a given conflicting hole, we calculate the average distance between the former and all the holes of each cluster associated to the base poses able to reach this conflicting hole. The lower the average distance, the closer the hole is to a given cluster. We therefore look for the smallest average distance and associate the considered hole to the corresponding cluster. This process is repeated for all the conflicting holes in order to make a partition.

The distance $d_{h_1 \to h_2}$ between two given holes h_1 and h_2 also takes into account the orientation between the two along the z axis, in order to consider the fact that two holes on different sides of a part are harder to reach. $d_{h_1 \to h_2}$ is defined as follows, where l is a characteristic length set to $1m$:

$$d_{h_1 \to h_2} = d_{h_2 \to h_1} = \sqrt{x^2 + y^2 + (l\theta)^2} \tag{5.10}$$

5.3.4.3 K-medoids and covering algorithm applied on the initial base poses sets

We developed three way of selecting the base poses thus far, from a clustering algorithm, or by applying one or multiple Set Cover algorithms to an already computed set of base poses. While we saw that the clustering algorithm lead to quite fluctuating results, we also noticed that the use of covering was time-consuming, due to the generation of initial base poses which still takes around $150s$ for a 240 holes part. Because of these results, we try to implement another variant.

While the generation algorithm requires a lot of time to run, it is important to mention that only the part where we check if the base poses can reach other holes is time-consuming. Indeed, the time needed to generate 240 base poses, with each of which able to reach one hole is far less important, as this process only takes around $5s$. For this reason, we try to pre-select an amount of base poses from which we will actually check if they can reach other holes, instead of trying it out for all of them.

In order to do this, we could for example randomly pre-select some base poses from the generated set. This would not be a suitable solution, as randomness does not guarantee that the base poses

are well scattered around the part and will effectively be able to reach all of the holes. Instead, we aim at selecting some base poses evenly scattered around the part.

Therefore, after studying ways of doing so, we came up with the idea of making different clusters of base poses, from which we could pre-select one or several base poses, before checking if they can reach the other holes of the part. Thankfully, such algorithms already exist. For example, a K-means algorithm can cluster a set of points depending on their coordinates. However, this method would raise 2 issues:

- First of all, this method only would consider the coordinates of each base pose, while we saw that their orientation is also an noteworthy attribute.
- Then, once the clusters are done, the K-means method also returns the center point of each cluster, which does not belong to the initial set of data.

While we could easily cope with these issues by not taking the orientation into account and selecting the closest base poses to the returned center points, we tried to look for a more suitable approach. We then came across a variant of the K-means method, which is called the K-medoids. This works in a similar way, but can work directly from a distance matrix. Besides, the center points returned (the medoids) are part of the initial entry set. We could therefore use this clustering approach by feeding it with a distance matrix taking into account the position as well as the orientation of the base poses.

We could once again use the exact distance between the base poses in order to build this distance matrix. This is however a costly process which is not necessary here. Indeed, we can afford to not consider the collisions, as our goal is to cluster the base poses around several medoids and not look for a shortest path. A way to evaluate a distance between two base poses while considering the orientation of the robot is to use the Reeds-Shepp [9] distance. Given the maximum turn radius of our robot, this distance is obtained from a trajectory taking into account the curves and the straight lines needed to connect two points. This distance is a variant of the Dubins distance [10], which did not take into account the possibility of reversing. As the Reeds-Shepp distance is fairly common in the path planning world, LAAS already has a tool allowing to evaluate such a distance between 2 given base poses.

We can therefore compute a Reeds-Shepp distance matrix between the base poses (without considering q_0). From there, we can apply the K-medoids algorithm in order to cluster the base poses. Although not mentioned in a paper, the algorithm we use was developed by Timo Erkkilä, Antti Lehmussola, Kornel Kiełczewski and Zane Dufour and is part of the scikit-learn-extra Python module. The algorithm has a $\mathcal{O}(n^2)$ complexity, where n is the size of the input matrix. Firstly, we test the algorithm on an initial set of base poses to see if the clustering works properly. We ask the algorithm to find 6 clusters among the 240 base poses given as an input. Figure 5.15 below shows the results.

The scene is viewed from on top, where each base pose is represented by a circle and an arrow pointing towards where the body of the robot is facing. The part is represented by the black rectangle. The depot, represented by a black cross, is barely visible, at the bottom of the figure. As for the clusters, each color represents one of them, with the blacked out base pose being the medoid of the said cluster. We can see that the clusters are well separated and distinct from

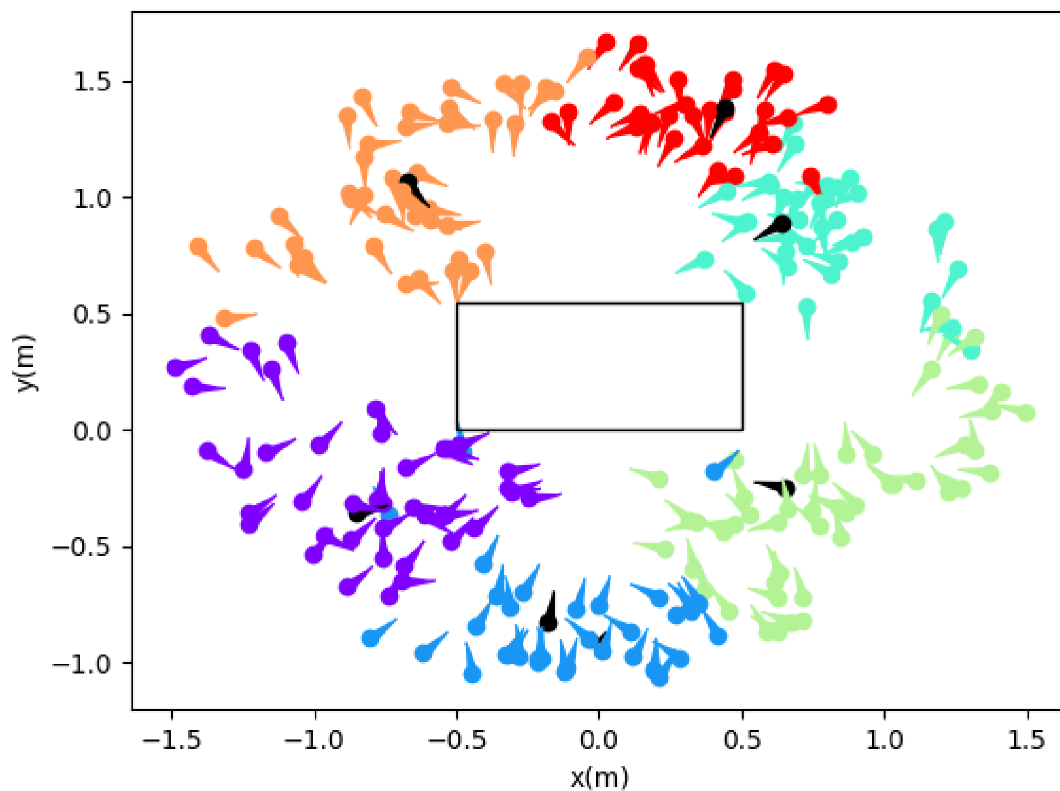


Figure 5.15: Overview of the base poses clusters and the part

each other. The position of each medoid is also suitable and seem to be at the barycentre of each cluster, as expected. The medoids are evenly scattered around the part, which was our initial aim. *Note:* some base poses seem to be far away from their corresponding cluster. This is because we used the Reeds-Shepp distance which is obviously different from the Euclidean distance, the later not taking the angles and actual trajectories into account.

We tried to reach all the holes from the $K_{medoids}$ chosen base poses, but we never managed to get a full covering of all the holes. We tried to iterate the process with different values of $K_{medoids}$, with little success. For this reason, we tried to add more base poses to our pre-selection by considering $N_{neighbours}$ around each medoid. In order to do so, we can for example select the closest base poses to each medoid. Again, we could either consider the Euclidean distances or the Reeds-Shepp ones. As we want to introduce more variety in our base poses selection, the Euclidean distance is not a suitable approach. Instead, we know that the closest base poses according to the Reeds-Shepp distances will be spatially further than the Euclidean ones. Furthermore, we also consider an angle threshold, in order not to consider base poses with a too similar orientation as the medoid. Figure 5.16 below shows the results, with $K = 6$ medoids and $N_{neighbours} = 5$ neighbours for each medoid.

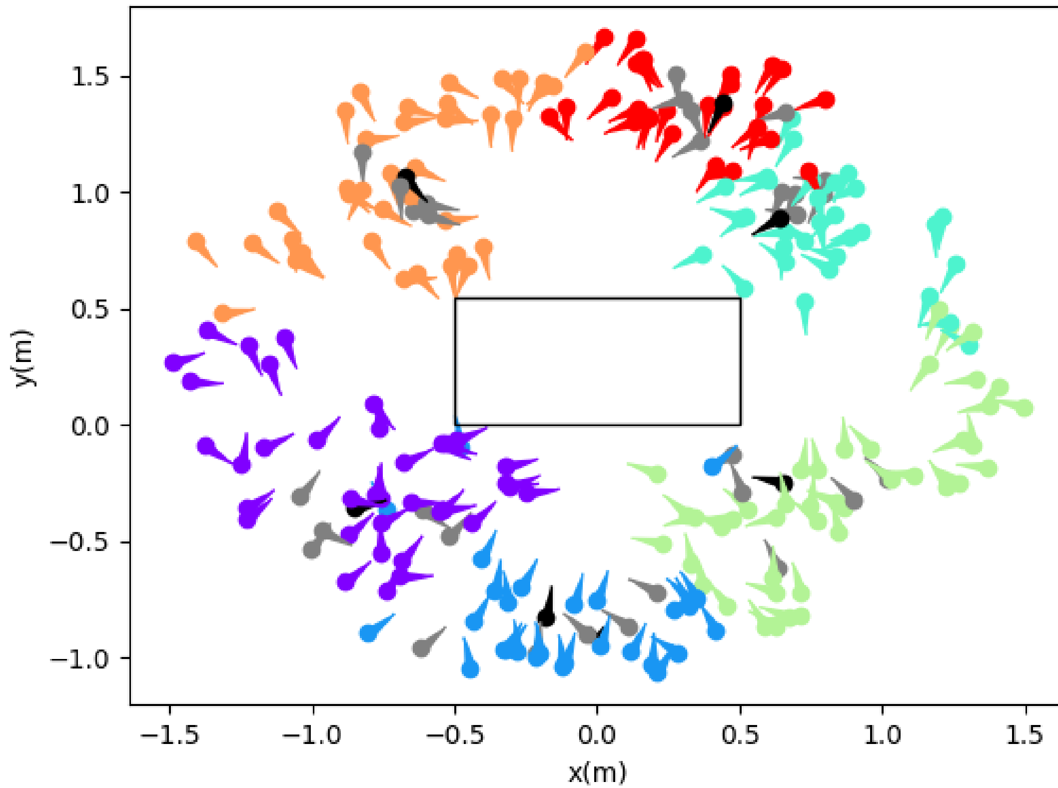


Figure 5.16: Overview of the base poses clusters and the part, with neighbours

The figure above is similar to the previous one, where greyed out base poses (the neighbours) have been added. We can see that they are well scattered around the space. We therefore managed to pre-select $K_{medoids} \times (N_{neighbours} + 1)$ base poses located all around the part. From there, we can apply the second part of the generation algorithm where we check whether each chosen base pose can reach the holes of the part. We use a threshold value similarly to what was described in section 5.3.4.1. Once a given amount of base poses is pre-selected, we can apply the methods described in section 5.3.4.2 in order to select the ones to use in the final tour.

Although several tests were done for different parameters and several initial data sets, we decide for the sake of readability to only show the results of several tests done from a single initial set of 240 base poses (see table 5.2). We only used one run of the covering algorithm in order to select the base poses. The tests were carried out using different values of $K_{medoids}$ and $N_{neighbours}$.

The studied input parameters are K_m and N_n which respectively correspond to the values of $K_{medoids}$ and $N_{neighbours}$; B_{total} is the amount of pre-selected base poses, equal to $K_{medoids} \times (N_{neighbours} + 1)$. As for the output values, we consider the amount of selected base poses after a run of the covering algorithm (B_{select}), as well as the computation time. First of all, we notice that small values of $K_{medoids}$ will lead to bad results regarding the selected base poses, regardless of the value of $N_{neighbours}$. This is explained because we are simply not looking at enough areas around the part in order to reach all the holes. Similarly, not exploring the neighbours will

almost always lead to a failure of the covering, for low values of $K_{medoids}$. Indeed, in that case, there is generally not enough base poses to look at. This approach starts to become more effective when $K_{medoids}$ is high enough. Nevertheless, the amount of returned base poses still fluctuates, even for high values of both input parameters.

K_m	N_n	B_{total}	B_{select}	t(s)
2	2	6	0	/
2	4	10	0	/
2	6	14	0	/
2	8	18	0	/
2	10	22	0	/
4	0	4	0	/
4	2	12	0	/
4	4	20	0	/
4	6	28	0	/
4	8	36	0	/
4	10	44	6	40.8
4	12	52	6	46.9
4	14	60	6	53.9
6	0	6	5	5.0
6	2	18	5	18.5
6	4	30	5	28.5
6	6	42	5	38.5
6	8	54	5	48.4
6	10	66	5	57.6
6	12	78	5	67.3
6	14	90	5	77.2
8	0	8	0	/
8	2	24	6	24.7
8	4	40	5	37.8
8	6	56	5	52.3
8	8	72	5	66.7
8	10	88	4	80.6
8	12	104	4	93.9
10	0	10	5	8.6
10	2	30	5	28.7
10	4	50	4	44.6
10	6	70	5	60.7

K_m	N_n	B_{total}	B_{select}	t(s)
10	8	90	4	77.4
10	10	110	4	94.3
15	0	15	4	11.6
15	2	45	4	39.3
15	4	75	5	64.5
15	6	105	4	82.5
20	0	20	4	16.5
20	2	60	4	51.1
20	4	100	4	101.1
25	0	25	5	33.8
25	1	50	5	55.1
25	2	75	4	63.0
25	3	100	4	85.2
30	0	30	4	23.5
30	1	60	4	48.0
30	2	90	5	76.0
35	0	35	4	27.2
35	1	70	4	55.2
35	2	105	4	84.9
40	0	40	5	41.9
40	1	80	4	66.2
45	0	45	4	36.6
45	1	90	4	74.8
50	0	50	4	40.5
50	1	100	4	82.0
55	0	55	4	44.3
55	1	110	4	103.5
60	0	60	4	49.6
65	0	65	5	66.1
70	0	70	4	57.6
75	0	75	4	60.6
80	0	80	4	66.2

Table 5.2: K-medoids algorithm results applied on an initial base poses set, for different values of $K_{medoids}$ and $N_{neighbours}$

Since tests done on a single generated set are not enough to determine appropriate values for $K_{medoids}$ and $N_{neighbours}$, we repeated the testing on different starting sets. Although the outcome confirmed what we described above, we could not identify a suitable range of $K_{medoids}$ and $N_{neighbours}$ leading to a stable result of 4 base poses. Instead, we noticed that the results fluctuate a lot depending on the starting set, similarly to the clustering method developed at

LAAS. It however appears that values of $K_{medoids}$ between 25 and 30 lead to a good compromise between the results and the computation times. We will set the value of $K_{medoids}$ to 30. As for $N_{neighbours}$, we saw that setting this value to 0 does not give good result. Since incrementing this value highly increases the computation time, we will set $N_{neighbours}$ to 1, which will help to improve the results of the 30 medoids.

A major drawback regarding this method, other than the fact that the results do fluctuate a lot, is that the choice of values for $K_{medoids}$ and $N_{neighbours}$ depend a lot on the part we study. Indeed, while theses value of $(30, 1)$ seem suitable for a part with 240 holes, we do not know which values to use on a part with 20 holes for example. Furthermore, a lack of robustness also appears, as increasing the amount of neighbours for a given amount of medoids do not always guarantee us to get a smaller or equal amount of selected base poses. This is because of the randomness of the generation algorithm and means that this method barely manages to reach all the holes with the least amount of base poses.

As for the other methods mentioned in this section, further testing is needed in order to evaluate their true efficiency time-wise and cost-wise. This is the aim of the following section.

5.3.5 Comparative study

We implemented 3 main methods aimed at selecting a set of base poses in order to further generate more arm configurations. We now have to compare them with the methods previously implemented in order to see which one ends up being a more suitable solution. Section 5.3.5.1 deals with comparisons on the 20 holes part, while section 5.3.5.2 focuses on the full part.

5.3.5.1 Comparative study on the 20 holes part

First, we test and compare all the algorithms on only 20 holes of the part. In order to do so, we compute average results for the algorithms and methods described in section 5.3, and compare them with the already computed values from section 5.2. The results are shown in table 5.3 below. We also study the influence of the main parameters linked to each method. For the sake of readability, some specific parameter values which were already tuned are not reminded in this table. As for the results, we present the total cost of the returned tour, as well as how it is split up between the base poses and the arms movements.

Firstly, we remind the results regarding the 3 methods mentioned in section 5.2. The exact, approximate and iterative methods were respectively described in sections 5.2.2, 5.2.3 and 5.2.4. As seen previously, both exact and iterative are very expensive time-wise but give better costs than the approximate method. Also, increasing the amount of base poses generated per hole (N) help to reduce the cost found by all 3 methods, while also increasing the computation times.

New methods had to be implemented in order to reduce the computation times on bigger parts. Indeed, while the approximate method seems to be a suitable approach in this case, it is too time-consuming for bigger scale problems. These 3 other methods try to select the base poses to use in the final tour, before adding more arm configurations and solving one or several GTSPs with less nodes than what was previously done and therefore faster computation times. Furthermore,

Method	Parameter 1	Parameter 2	Weight	Bases	Arms	t(s)
Exact	$N = 1$	/	60.9	45.1	15.8	91.8
Exact	$N = 3$	/	53.6	40.2	13.3	270.2
Approximate	$N = 1$	/	62.8	42.9	19.9	15.4
Approximate	$N = 3$	/	55.5	39.2	16.3	30.3
Iterative	$N = 1$	/	63.1	44.7	18.5	144.1
Iterative	$N = 3$	/	54.4	40.2	14.2	161.9
Clustering	$N_{tries} = 20$	/	71.0	57.0	14.0	25.0
Covering	$N_{runs} = 1$	$Partition = False$	66.7	51.2	15.4	14.0
Medoids + Covering	$K_{medoids} = 5$	$N_{neighbours} = 1$	70.5	54.9	15.6	13.6
Covering	$N_{runs} = 1$	$Partition = True$	66.8	51.2	15.6	24.4
Medoids + Covering	$K_{medoids} = 5$	$N_{neighbours} = 1$	68.1	53.8	14.4	23.6
Covering	$N_{runs} = 5$	$Partition = False$	58.8	41.9	16.9	23.3
Medoids + Covering	$K_{medoids} = 5$	$N_{neighbours} = 1$	62.4	48.3	14.1	19.6
Covering	$N_{runs} = 5$	$Partition = True$	58.6	41.8	16.8	32.9
Medoids + Covering	$K_{medoids} = 5$	$N_{neighbours} = 1$	61.1	46.3	14.8	30.2

Table 5.3: Comparison of all methods and their variants for a 20 holes problem

the addition of new arm poses should reduce the cost related to the arms and could potentially balance out the losses due to the fact that the base poses are pre-selected.

The clustering algorithm already implemented at LAAS gives higher costs than other methods due to the heuristic used for the base poses selection. Indeed, further cost analysis show that while the cost related to the arms is similar to other methods, the cost linked to the base movements is much higher. This higher cost is most likely due to the use of more base poses than the other methods. Indeed, as seen in section 5.3.3, the amount of returned base poses fluctuates a lot with this method.

The covering approach can either start with the usual base poses generation algorithm, or with a medoids approach, which is supposedly less time-consuming. Consequently, tests done with the Medoids + Covering method use the same covering parameters (N_{runs} and partitioning) as the ones listed in the above row within the tables. Regarding the covering approach using the standard base poses generation algorithm, the cost is greatly reduced when N_{runs} increases, since it allows visiting more base poses and therefore lead to a potential cost reduction. Also, the partitioning of the holes does not have a noticeable impact on the total cost, which shows that the partitioning heuristic mentioned is a suitable approach. This variant allows to solve multiple GTSPs and should therefore reduce the computation times. It is however not the case here. Indeed, such a small instance does not have a major effect on solver computation times, and it actually takes longer to solve multiple GTSPs because the solver spends time starting up at each iteration.

Finally, we analyse the results regarding the covering approach starting from a generation involving medoids. While we previously set $K_{medoids}$ and $N_{neighbours}$ to respectively 30 and 1, this approach cannot be done for such a small instance. Indeed, we only generate 1 initial base per hole, which leads to a total of 20 base poses. We can therefore not select 60 base poses out of this set and adapted the values of $K_{medoids}$ and $N_{neighbours}$ to this smaller instance. The

computation times are faster than the corresponding tests using the covering method and the generation algorithm, because the medoids approach do not consider all the base poses when trying to reach the holes of the part. For this reason, the total cost is higher than when using the covering algorithm, because not all the base poses are considered when trying to cover the holes. The value of N_{runs} and the choice of making a partition have similar effects as mentioned previously. We notice that increasing N_{runs} has a non-deterministic impact on the costs related to the arms, and can either increase or lower it. This is because the choice of the base poses is done independently of the costs related to the arms when $N_{runs} > 1$ (they are set to 0 in the cost matrix, see 5.14).

To conclude, most of the methods tested on a 20 hole problem give us similar results, whether it be time-wise or cost-wise. While some methods are definitely not usable due to their high computation times, it is hard to tell apart the best from this first series of testing. The approximate method with $N = 3$ base poses generated per hole gives us the best costs, but we already know that it cannot be applied on bigger problems due to its high amount of nodes and therefore high computation times. However, some other methods show close results cost-wise but should be faster on bigger instances. Overall, the clustering, covering and medoid methods show a reduced cost related to the arms thanks to the generation of additional arm configurations but a higher cost related to the movements of the base due to the selection methods used.

5.3.5.2 Comparative study on the 240 holes part

After studying the performance of several methods on a small 20 holes instance, we test them on a much bigger scale problem with 240 holes. While the time efficiency was similar on the 20 holes problem, some gaps should start to appear on this instance and help identify the best(s) solution(s).

Due to already very high computation times and memory issues, we do not test the exact and iterative methods on this instance. Similarly to what we did in section 5.3.5.1, the results from average tests are shown in table 5.4 below.

Method	Parameter 1	Parameter 2	Weight	Bases	Arms	t(s)
Approximate	$N = 1$	/	142.6	86.5	56.1	1943.6
Clustering	$N_{tries} = 20$	/	157.9	90.9	67.0	152.2
Covering	$N_{runs} = 1$	$Partition = False$	162.4	94.5	67.9	382.4
Medoids + Covering	$K_{medoids} = 30$	$N_{neighbours} = 1$	151.3	91.4	60.0	285.9
Covering	$N_{runs} = 1$	$Partition = True$	163.3	94.9	68.4	249.5
Medoids + Covering	$K_{medoids} = 30$	$N_{neighbours} = 1$	169.4	94.8	74.6	161.0
Covering	$N_{runs} = 5$	$Partition = False$	153.5	83.9	69.7	418.3
Medoids + Covering	$K_{medoids} = 30$	$N_{neighbours} = 1$	152.4	82.0	70.4	319.1
Covering	$N_{runs} = 5$	$Partition = True$	153.4	84.4	69.0	310.6
Medoids + Covering	$K_{medoids} = 30$	$N_{neighbours} = 1$	150.5	84.3	66.2	208.0

Table 5.4: Comparison of all methods and their variants for a 240 holes problem

First of all, we can see that the approximate method gives the best results cost-wise. Indeed,

this method has a much lower cost related to the movements of the arms than its counterparts. This can be explained by two reasons:

- On the one hand, a visual analysis of the tours given by this method showed that the movements related to the arms were already very smooth and well optimised, which was not the case when considering 20 of the holes. This explains why the other methods, all using the same algorithms to generate more arm configurations do not improve this cost.
- On the other hand, the reason why the costs found by the new methods end up being worse is because these methods optimise the movements of the arms and the base poses independently. However, the choice of a base pose has consequences for the movements of the arms, mainly related to its extension and retraction when reaching the part. Indeed, we see in some cases that the arm needs to maneuver a lot in order to avoid collisions with the part. This could be due to the base being too close to the part for example. This also explains why the covering methods tends to find lower base costs (with the proper parameters) than the approximate one, as both methods use the same initial graph but the covering ones optimise the base pose independently, leading to better costs linked to the base

While the approximate method has much better costs, it also takes much longer to compute, since it takes into account both the base poses and the arms at the same time, and is therefore not well adapted to large-scale problems.

Thanks to its base selection heuristic, the clustering approach is the fastest one. It also ends up being better than the covering approaches, when these run the covering algorithm only once. However, we know that this method has quite fluctuating results, as it will not always find the lowest possible amount of base poses. Furthermore, robustness issues could appear in some specific cases. Indeed, the part we studied had many holes close to each other and the choice of the starting hole to generate a cluster did not have much of an impact. However, on a part with isolated holes, generating a cluster from a hole located far from its neighbours will most likely lead to an isolated base pose which cannot reach a high amount of holes.

The covering following a "standard" base generation and the medoids + covering approaches both show poor results when doing only a single run, since they do not try to optimise the cost related to the base poses, but simply look for the smallest covering set. Since several sets can usually cover our holes using the same amount of base poses, running this method several times before choosing the best base poses help to reduce the cost a lot.

Moreover, unlike with the 20 holes part, the partitioning of the holes helps to lower the computation times a lot, as several smaller problems take shorter to solve than a single big one. We also notice that the costs related to the arms stay similar most of the time

Furthermore, the use of the medoids + covering approach help to reduce the computation times linked to the generation algorithm used otherwise. However, due to a lack of robustness, the costs returned by this method tend to fluctuate a lot. Indeed, it sometimes gives better results than the standard covering approach, but also becomes worse in other cases. Additionally, the parameters of this approach depend on the problem a lot, as a poor choice of $K_{medoids}$ and $N_{neighbours}$ could lead to a failure of the covering. It is almost impossible to determine these parameters without knowing how the part will look like. Another robustness issue is the fact

that the amount of covering sets of relatively small size found by this method is low. Indeed, running the covering algorithm several times on a graph computed by this method quickly tends to find sets of size 6 or 7 which never happens with the standard approach. While this is not a problem in our case, the addition of more constraints (for example visibility constraints or new obstacles around the part) could lead to a drastic cost increase.

Selecting the best method from these tests is not an easy task. Even though the costs are relatively similar but the computation times are in favour of either the clustering or the medoids approach, both methods show potential robustness issues which could be a problem on specific parts. As for the standard deviations on the costs (not shown here), they are relatively close for all three methods, around 15. This deviation is quite high and shows how the random initial base poses set can influence on the results.

The choice of the most suitable method depends on how this algorithm would be used in an industrial approach. Would it only need to be run once in order to save the best tour and apply it repeatedly, or would it be run every time a part needs to be deburred? In the former case, the most robust approach is definitely suitable, as the computation times are not a key aspect. For this reason, the covering approach with several runs would be the chosen one. In the latter case, we would need to evaluate the losses linked to each method regarding the computation times and the deburring time, in order to see if it is worth spending more time computing a solution or having faster computation times with less robustness and potential failures. For example, some approaches could fail at reaching all the holes, or do so with a high amount of base poses, both leading to higher deburring time and increasing the probability for the robot to get lost and make the whole process last even longer by trying to retrieve its route.

As a conclusion, all three of the approaches involving a base selection prior to an optimisation of the movements of the arms are much faster than the approximate method involving a high amount of nodes, but it is hard to tell which one is the best as they all have their specific strengths and weaknesses. The choice of the most suitable approach will eventually depend on which context the algorithm will be used within.

5.3.6 Study of the solving approach using Dijkstra's algorithm

The algorithm described in [4] can be adapted to our problem. Indeed, this solution can only be used with a fixed base pose and can thus be tested if we do a partitioning of the holes. After solving a TSP for the base poses, we decide to solve multiple GTSPs for each cluster in our partitioning approach. Instead, we could use the method involving a predetermined tour between the holes, and then apply Dijkstra's algorithm to choose which arm configuration to use, while following the tour planned initially.

In order to test this other approach, we use the series of tests done previously involving the covering algorithm with 5 runs and a partitioning of the holes as a comparison. We only compare the solving time for the arm configurations, as everything else remains the same between both approaches. The goal is indeed to find out whether the use of multiple GTSPs is better than the use of multiple Dijkstra's algorithms applied on an already scheduled tour among the holes.

Table 5.5 below shows the results, where both solving algorithms were run using the same amount of arm configurations. The cost is drastically higher when using the Dijkstra approach.

While the cost related to the base poses is very similar (supposed to be identical but fluctuation appears when using the path planning algorithms), the cost related to the arms is much worse with the Dijkstra method. This is explained because the tour is already pre-computed and the best choices among the arm configurations cannot be made generally, but only within the already chosen deburring order.

Method	Cost	Bases	Arms	t(s)
multiple GTSPs	153.4	84.4	69.0	36.8
multiple TSPs + Dijkstras	185.1	82.4	102.7	30.3

Table 5.5: Comparison between the GTSP and Dijkstra algorithms

As for the computation times, it is important to mentions that they are a bit misleading. Indeed, the TSP algorithm we used to plan a tour of the holes takes some time to start up and ends up consuming a lot of time. However, once the tour is planned, the Dijkstra approach would be much quicker than the use of GTSPs, and would only take a few seconds to find the best path.

Since the costs found by the second approach are not good, we try to add more arm configurations to see if this approach could become an alternative to consider. We generate twice as many arm configurations as with the GTSPs approach. This greatly improved the costs related to the movements of the arms, but the GTSPs approach still gives about 20% better results. The computation times remained almost the same, showing the main advantage of the Dijkstra approach, where adding nodes does not have a significant impact on the computation times. With three times as many configurations, the costs do get better, but the GTSP approach still has about 13% better results. Also, the solving time increases by about $5s$. However, the generation times increase significantly, going from about $4s$ to $33s$ when the amount of configurations is increased threefold.

Therefore, the Dijkstra approach is not a suitable alternative in our case, because of overall higher computation times due to the necessity of generating additional configurations. It is however not useless and could end up being a better approach in some specific cases. For example, in the case where each cluster would have a high amount of holes, the GTSP approach could become much more time-consuming. Indeed, the study done in [4] showed that using a GTSP on a 200 holes cluster was not a suitable solution compared to the Dijkstra approach.

Chapter 6

Critical Viewpoint and Conclusions

We developed and studied several solutions to the task sequencing problem with a moving base. Section 6.1 presents a critical viewpoint of what we achieved, while section 6.2 deal swith how this study could be used in the future, as well as several possible axes of improvement. Finally, section 6.3 delivers a more personal conclusion on the internship and its subject.

6.1 Critical viewpoint

All along the internship, we developed several methods and variants tackling task scheduling problems with the use of GTSPs. We managed to successfully improve the first methods we implemented - which would not even be usable on big scale problems - in order to reach feasible computation times. Furthermore, this was done without having a major impact on the costs. Indeed, the computation times linked to the approximate method were reduced by at least 5 times when considering other approaches, with a low impact on the costs (about 7% loss). We also managed to find a solution closer to the optimum that the one involving Dijkstra's algorithm [4], which is a contribution to the research in the robotics field. We did this by combining already existing algorithms to ours, while trying to optimise the ones we developed as much as possible.

However, we encountered several limiting aspects to our study. First of all, the testing showed that the results could fluctuate a lot depending on the initial generated set. This set being generated randomly, the actual position of the base poses is never properly optimised, as the best ones are simply chosen among a finite set. Furthermore, even though it was improved, the generation algorithm still accounts for about half the total computation times. Although alternatives were also studied, such as the K-medoids and clustering approaches, both methods supposedly have robustness drawbacks. These possible robustness issues were not encountered during our testing. This shows that the testing potentially did not reveal all the strengths and weaknesses of each method. Indeed, additional testing on different shapes of parts and different scattering of the holes must be done in order to ensure our solutions are actually viable for different problems and not only the ones we studied.

Moreover, all the solutions we implemented involve approximations regarding the movements of the arms. Indeed, once the tour is set, the movements are computed with no guarantee of good results. This shows the limit between our optimisations and what could possibly be done at the lower level of path planning algorithms. Finally, we could not compare our method to the already existing approach at LAAS, due to Python compatibility issues. However, Florent Lamiraux did a demonstration of it at the beginning of the internship, showing long computation times due to the TSP used, as well as high movement costs, due to the lack of configurations to choose from.

Additionally, I initially encountered some difficulties when familiarising with the generation and path planning tools developed at LAAS, as these have the prerequisite of understanding robotics well. Thankfully, my internship supervisor helped me understand these important notions in order to build the necessary algorithms. Some technical issues were also encountered and also took some time to be solved, especially because of the teleworking context.

6.2 Perspectives

These solutions we implemented will be used for future LAAS studies, as they can be seen as a prospective study and a survey between different methods. This internship will indeed help LAAS build what they think is the best approach in order to suit Airbus' needs as part of the ROB4FAM project. The results are encouraging and the next step is to test the methods on a real part instead of simply using a visual interface.

Although we initially planned to improve the path planning algorithms developed at LAAS during the second part of the internship, we instead decided to keep trying to improve our solutions in order to be able to tackle large-scale problems of more than 200 holes, which is usually the case in industrial applications.

Moreover, our algorithms can be further improved. While we mentioned the possible need of improving the path planning algorithms, leading to better costs, the computation times can also be improved. For example, the multiple GTSPs can be ran on distributed systems and therefore be executed simultaneously, since they are independent. More specific cost-related improvements can also be done. For example, we could implement a way of choosing the best starting holes for the clustering algorithm, or we could improve the base poses' selection of the covering algorithm by associating each base pose to a cost of use. Also, the K-medoids approach could be improved by implementing a way of finding its best parameters.

To conclude, we developed several methods which will be used in the future by LAAS and possibly by Airbus thanks to the common ROB4FAM laboratory. Although there is still room for improvement, our methods are an alternative approach to consider when comparing it with the already existing solutions developed in the robotics field.

6.3 Personal conclusion

As a personal conclusion, this internship helped me discover several new domains which I did not know before. Indeed, I was able to explore both the robotics and research fields, while also learning more about optimisation. Even though I had some initial trouble accustoming to these domains, I quickly gained in autonomy and managed to think about and develop solutions to the problem myself. Indeed, this project helped me realise that I can adapt the knowledge I learned at ENAC in order to explore new fields which I did not now deeply before. Conclusively, this internship helped me build the much needed final link between the academic and professional worlds.

Bibliography

[1] Florent Lamiraux and Joseph Mirabel. "Prehensile Manipulation Planning: Modelling, Algorithms and Implementation". In: *Rapport LAAS n°20298* (2020). URL: https://hal.laas.fr/hal-02995125.

[2] Shaohua YU, Jakob Puchinger, and Shudong Sun. "Two-echelon urban deliveries using autonomous vehicles". In: *Transportation Research Part E: Logistics and Transportation Review* 141 (Sept. 2020). URL: https://hal.archives-ouvertes.fr/hal-02877730.

[3] Sergey Alatartsev, Sebastian Stellmacher, and Frank Ortmeier. "Robotic Task Sequencing Problem: A Survey". In: *Journal of Intelligent & Robotic Systems* (Mar. 2015).

[4] Francisco Suárez-Ruiz, Teguh Santoso Lembono, and Quang-Cuong Pham. "RoboTSP - A Fast Solution to the Robotic Task Sequencing Problem". In: (2017). URL: https://arxiv.org/abs/1709.09343.

[5] S. L. Smith and F. Imeson. "GLNS: An Effective Large Neighborhood Search Heuristic for the Generalized Traveling Salesman Problem". In: *Computers & Operations Research* 87 (2017), pp. 1–19. URL: https://ece.uwaterloo.ca/~sl2smith/papers/2017COR-GLNS.pdf.

[6] Gilbert Laporte, Hélène Mercure, and Yves Nobert. "Generalized travelling salesman problem through n sets of nodes: the asymmetrical case". In: *Discrete Applied Mathematics* 18.2 (1987), pp. 185–197. URL: https://www.sciencedirect.com/science/article/pii/0166218X87900205.

[7] V. Chvatal. "A Greedy Heuristic for the Set-Covering Problem". In: *Mathematics of Operations Research* 4.3 (1979), pp. 233–235.

[8] Guangtun Zhu. "A New View of Classification in Astronomy with the Archetype Technique: An Astronomical Case of the NP-complete Set Cover Problem". In: (2016). URL: https://arxiv.org/abs/1606.07156.

[9] J. Reeds and L. Shepp. "Optimal Paths for a Car That Goes Both Forwards and Backwards". In: *Pacific Journal of Mathematics* 145 (1990), pp. 367–393.

[10] L. Dubins. "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents". In: *American Journal of Mathematics* 79 (1957), p. 497.

Appendices

Appendix A

Glossary

Configuration: A vector containing the position and orientation of all the joints of a robot at a given time. It can therefore entirely describe the pose of a robot at a given time.

Deburring: The action of removing small sharp edges appearing on a machined metal part.

Dijkstra's algorithm: An algorithm aimed at finding the shortest path in a weighted graph, between given starting and ending nodes.

Generalised Travelling Salesman Problem: A problem similar to the TSP, but were the vertices of the graph are partitioned into clusters and exactly one vertex per cluster must be visited.

Hamiltonian cycle: A visiting order of all the vertices of a graph, where every vertex is visited exactly once, returning to its starting vertex. In this report, we abusively denote it as a *tour*.

Joint: The movable articulation linking two rigid parts of a robot.

K-means clustering: A clustering method where each element of a given set is grouped with the cluster having the nearest centroid.

K-medoid clustering: A variant of the K-means method, where the centroids themselves (called the medoids) are elements belonging to the input set.

Partition: A partition of a set is a grouping of its elements into disjoint and non-empty subsets.

Travelling Salesman Problem: The problem of finding the shortest Hamiltonian cycle within a graph of weighted edges.

Appendix B

Acronyms

CNRS . . . . . French National Centre for Scientific Research
GLNS . . . . . *Unknown - never mentioned by the authors*
GTSP . . . . . Generalised Travelling Salesman Problem
GUI . . . . . . Graphical User Interface
HPP . . . . . . Humanoid Path Planner
LAAS . . . . . Laboratory of Analysis and Architecture of Systems
ROB4FAM . . Robots For the Future of Aircraft Manufacturing
ROC . . . . . . Operations Research, Combinatorial Optimisation and Constraints
SCP . . . . . . Set Cover Problem
TSP . . . . . . Travelling Salesman Problem

Appendix C

Deeper Algorithm Descriptions

C.1 Generation algorithm

Algorithm 5 below describes the base generation process.

The algorithm is split up in three parts. The first part (lines 1-12) tries to compute N sets of valid configurations for each hole, where a set contains three configurations: one with the drill about to enter the hole ($q_{pre\ grasp}$), one with the drill in the hole (q_{grasp}) and another one with Tiago's arm retracted (q_{base}). It is necessary to generate both a pre-grasp and a grasp configuration in order to create a translation movement between the two and therefore simulate the deburring of a hole. Each of these three configurations are computed using three similar functions:

- $genValidPreGraspConfig()$ generates a valid pre-grasp configuration for a given hole (line 6).
- $genValidGraspConfig(hole, q_{pre\ grasp})$ generates a valid grasp configuration for the same hole, from the pre-grasp configuration previously generated (line 7).
- $genValidBaseConfig(hole, q_{pre\ grasp})$ generates a valid base configuration (with the arm retracted) for the same hole, from the pre-grasp configuration previously generated (line 9). The configuration of the arm once retracted will always be the same and corresponds to the configuration it had in q_0.
- The three functions described above are already existing at LAAS. They work by generating an initial set of configurations and checking whether they are valid i.e. there are no collisions. If it is the case, one configuration is returned among the set of valid configurations generated.

If the two pre-grasp and grasp configurations are successfully generated, the base configuration is retrieved and the set is stored in the dictionary $D^{holes:configs}$ associating each hole to a list of configuration sets (lines 9 - 12). Furthermore, the hole is added to the dictionary $D^{configs:holes}$ associating each base configuration (q_{base} here) to the list of holes it can reach (line 11).

The second part of the algorithm (lines 13 - 21) checks whether the sets of configurations previously generated for a given hole could reach all the other holes of the part. This process is

Algorithm 5: Generate base poses

Input: $N, holes$
Output: $D^{holes:configs}, D^{configs:holes}$

1 **Initialize:** $D^{holes:configs} \leftarrow \emptyset, D^{holes:configs}_{temp} \leftarrow \emptyset, D^{configs:holes} \leftarrow \emptyset$
2 **foreach** $hole \in holes$ **do**
3     $D^{configs:holes}[hole] \leftarrow \emptyset$
4     $N_{added} \leftarrow 0$
5   **while** $N_{added} < N$ **do**
6         $q_{pre\ grasp} \leftarrow genValidPreGraspConfig(hole)$
7         $q_{grasp} \leftarrow genValidGraspConfig(hole, q_{pre\ grasp})$
8   **if** $q_{pre\ grasp} \neq None$ and $q_{grasp} \neq None$ **then**
9             $q_{base} \leftarrow genValidBaseConfig(hole, q_{pre\ grasp})$
10   add $(q_{pre\ grasp}, q_{grasp}, q_{base})$ to $D^{holes:configs}[hole]$
11   add $hole$ to $D^{configs:holes}[q_{base}]$
12             $N_{added} = N_{added} + 1$
13 **foreach** $hole \in holes$ **do**
14   **foreach** $(other_hole, configs) \in D^{holes:configs}, other_hole \neq hole$ **do**
15   **foreach** $config_{array} \in configs$ **do**
16             $q_{base} \leftarrow config_{array}[2]$
17             $q_{pre\ grasp} \leftarrow genValidFixedPreGraspConfig(hole, q_{base})$
18             $q_{grasp} \leftarrow genValidGraspConfig(hole, q_{pre\ grasp})$
19   **if** $q_{pre\ grasp} \neq None$ and $q_{grasp} \neq None$ **then**
20   add $(q_{pre\ grasp}, q_{grasp}, q_{base})$ to $D^{holes:configs}_{temp}[hole]$
21   add $hole$ to $D^{configs:holes}[q_{base}]$
22 **foreach** $(hole, configs) \in D^{holes:configs}_{temp}$ **do**
23   **foreach** $config_{array} \in configs$ **do**
24   add $config_{array}$ to $D^{holes:configs}[hole]$
25 **return** $D^{holes:configs}, D^{configs:holes}$

repeated for all the holes. For all the base configurations previously generated for a given hole the algorithm tries to generate a valid pre-grasp configuration reaching every other hole of the part (lines 15-17). $genValidFixedPreGraspConfig()$ is similar to $genValidPreGraspConfig()$ but will try to compute a pre-grasp configuration with the robot not moving its base from where it was in q_{base}. Then, a grasp configuration is generated (line 18).

If the 2 configurations are successfully generated, the set $(q_{pre\ grasp}, q_{grasp}, q_{base})$ is added to a temporary dictionary $D^{holes:configs}_{temp}$ associating the current hole to a list of configuration sets (lines 19 - 21). This new dictionary, although similar to $D^{holes:configs}$, needs to be created in order to avoid unnecessarily going through the same base configuration multiple times. Finally, the hole is added to the dictionary $D^{configs:holes}$ associating each base configuration (q_{base} here) to the list of holes it can reach (line 21).

The last part of the algorithm copies the content of $D^{holes:configs}_{temp}$ into $D^{holes:configs}$ (lines 22 - 24).

Two dictionaries are returned, $D^{configs:holes}$ associates each base pose to the list of holes it can reach, $D^{holes:configs}$, on the other hand associates each hole to the list of configuration sets $(q_{pre\ grasp}, q_{grasp}, q_{base})$ that can reach it. Both dictionaries will be useful in order to solve the corresponding GTSP, as they are defining the graph.

C.2 Exact cost matrix building process

Algorithm 6 below outlines the building process of the cost and path matrices linked to the exact method:

Algorithm 6: Build the cost matrix

Input: $D^{holes:configs}, D^{configs:holes}, configs, nodes$
Output: $costs_matrix, paths_matrix$

```
1  Initialize: D^{configs:distances} ← ∅, costs_matrix ← ∅, paths_matrix ← ∅,
2  N ← LEN[nodes]
3  base_costs, base_paths ← ComputeBaseMatrix(configs)
4  foreach base ∈ configs \ {q_0} do
5      arm_costs, arm_paths ← ComputeArmMatrix(base, D^{configs:holes})
6      add arm_costs, arm_paths to D^{configs:distances}
7  foreach i ← 0 to N − 1 do
8      foreach j ← i + 1 to N − 1 do
9          node_1 ← nodes[i]
10         node_2 ← nodes[j]
11         if q_1 = q_0 then
12             cost ← 2.5 × cost_{q_0→q_2} + cost_{q_2→h_2|q_2}
13             path ← path_{q_0→q_2} + path_{q_2→h_2|q_2}
14         else if q_2 = q_0 then
15             cost ← cost_{h_1|q_1→q_2} + 2.5 × cost_{q_1→q_0}
16             path ← path_{h_1|q_1→q_2} + path_{q_1→q_0}
17         else if q_1 = q_2 then
18             cost ← cost_{h_1|q_1→h_2|q_1}
19             path ← path_{h_1|q_1→h_2|q_1}
20         else
21             cost ← cost_{h_1|q_1→q_1} + 2.5 × cost_{q_1→q_2} + K_p + cost_{q_2→h_2|q_2}
22             path ← path_{h_1→q_1} + path_{q_1→q_2} + path_{q_2→h_2}
23         UpdateMatrix(costs_matrix, paths_matrix, cost, path)
24 return costs_matrix, paths_matrix
```

The input of the algorithm is the dictionaries associating the holes and the configurations, as well as the list of base poses including q_0 (the depot) and the list of nodes. The nodes correspond to the rows and the columns of the costs and paths matrices. As several configurations can be associated with each hole, a given node corresponds to a couple (q, h) and thus describes the robot deburring the hole h in configuration q. *Note:* the node associated to q_0 is $(q_0, None)$.

The computations are split up and associated in one final matrix. Line 3 computes the two

matrices associated to the base poses. Thus only the movement of the base of the robot are taken into account during this step. Then (lines 4 - 6), two matrices corresponding to the movement of the arms are computed for each base. These matrices therefore correspond to all the possible movements of the arm within the same base pose, which includes moving between all the holes associated to this pose, as well as extending and retracting the arm between the holes and the base configuration.

The contents of the matrix computation algorithms are not detailed here for the sake of readability. They use LAAS algorithms to build paths between the input configurations. The resulting paths and lengths are stored in their corresponding matrices. If the path planning algorithms fail to compute a path, it is set to $None$ and its length is set to an artificially high value of 10^8.

From all the matrices computed, (lines 7 - 23), the costs and the paths are then associated in order to build the costs and paths between all the nodes. Since the problem is symmetrical, only half of both the costs and paths matrices are computed. As mentioned earlier, the cost related to the movement of the base is weighted if it is necessary to change the base pose between two nodes (line 21).

We note $h_1|q_1 \rightarrow q_1$ the movement associated to the robot moving from deburring hole h_1 from base configuration q_1 to its position with the arm retaracted in q_1. This corresponds to retracting its arm from h_1 to the base pose q_1. Similalry, we note $h_1|q_1 \rightarrow h_2|q_1$ the movement corresponding to going from hole h_1 to hole h_2 while staying in the same base pose (q_1). $q_1 \rightarrow q_2$ is the movement of the base from q_1 to q_2.

The cost and path are then put in the two final matrices (line 23). The output of the algorithm is the two matrices built: one containing all the costs associated to the movement between each of the nodes, as well as the corresponding paths matrix, which stores these movements.

C.3 Approximate cost matrix building process

Algorithm 7 below outlines the building process of the cost and path matrices linked to the approximate method:

Algorithm 7: Build the approximate cost matrix

Input: $D^{holes:configs}, D^{configs:holes}, configs, nodes$
Output: $costs_matrix^{approx}, base_paths$
1 **Initialize:** $D^{configs:distances} \leftarrow \emptyset, costs_matrix^{approx} \leftarrow \emptyset, N \leftarrow$LEN$[nodes]$
2 $base_costs, base_paths \leftarrow ComputeBaseMatrix(configs)$
3 **foreach** $base \in configs \setminus \{q_0\}$ **do**
4 $\quad arm_costs^{approx} \leftarrow ComputeApproxArmMatrix(base, D^{configs:holes})$
5 $\quad$ add arm_costs^{approx} to $D^{configs:distances}$
6 **foreach** $i \leftarrow 0$ *to* $N-1$ **do**
7 $\quad$ **foreach** $j \leftarrow i+1$ *to* $N-1$ **do**
8 $\quad\quad node_1 \leftarrow nodes[i]$
9 $\quad\quad node_2 \leftarrow nodes[j]$
10 $\quad\quad$ **if** $q_1 = q_0$ **then**
11 $\quad\quad\quad cost^{approx} \leftarrow 2.5 \times cost_{q_0 \to q_2} + cost^{approx}_{q_2 \to h_2|q_2}$
12 $\quad\quad$ **else if** $q_2 = q_0$ **then**
13 $\quad\quad\quad cost^{approx} \leftarrow cost^{approx}_{h_1|q_1 \to q_2} + 2.5 \times cost_{q_1 \to q_0}$
14 $\quad\quad$ **else if** $q_1 = q_2$ **then**
15 $\quad\quad\quad cost^{approx} \leftarrow cost^{approx}_{h_1|q_1 \to h_2|q_1}$
16 $\quad\quad$ **else**
17 $\quad\quad\quad cost^{approx} \leftarrow cost^{approx}_{h_1|q_1 \to q_1} + 2.5 \times cost_{q_1 \to q_2} + K_p + cost^{approx}_{q_2 \to h_2|q_2}$
18 $\quad\quad UpdateMatrix(costs_matrix^{approx}, cost^{approx})$
19 **return** $costs_matrix^{approx}, base_paths$

The process is very close to the one described in algorithm 6, with the major change being the fact that the costs related to the movements of the arms are now approximated (line 5). Also, the paths related to these movements are obviously not generated and the paths matrix is not built in this method. Only the paths matrix related to the base poses is returned.

C.4 Find the best cluster around a given hole

Algorithm 8 below outlines the process of finding the best cluster around a given hole.

Algorithm 8: LAAS algorithm to find one cluster around a given hole

---

Input: $h_{start}, holes, remaining_holes, N_{tries}$
Output: $best_cluster$

1. **Initialize:** $best_cluster \leftarrow \emptyset$
2. **foreach** $i \leftarrow 1$ *to* N_{tries} **do**
3. $\quad qp_i \leftarrow genValidPreGraspConfig(h_{start})$
4. $\quad q_i \leftarrow genValidGraspConfig(h_{start}, qp_i)$
5. $\quad current_cluster \leftarrow [(h_{start}, qp_i, q_i)]$
6. $\quad$ get q_{base} from $current_cluster$
7. $\quad$ **foreach** $other_hole \in holes, other_hole \neq h_{start}$ **do**
8. $\quad\quad q_p \leftarrow genValidFixedPreGraspConfig(other_hole, q_{base})$
9. $\quad\quad q \leftarrow genValidGraspConfig(other_hole, q_p)$
10. $\quad\quad$ **if** $q_p \neq None$ and $q \neq None$ **then**
11. $\quad\quad\quad$ add $(other_hole, qp, q)$ to $current_cluster$
12. $\quad$ **if** LEN($current_cluster$) > LEN($best_cluster$) **then**
13. $\quad\quad best_cluster \leftarrow current_cluster$
14. **return** $best_cluster$

The algorithm in charge of looking for a cluster from a randomly selected starting hole (h_{start}) works as follows. First, a base pose is associated with this hole (lines 3 - 6). Then, we check how many holes we can reach from the same base pose (lines 7 - 11). The process is repeated with N_{tries} different base poses (line 2). The cluster covering the maximum amount of holes is then returned.

Name or the Intern: Réot Antoine

Course: IENAC18 AVI

Company: LAAS CNRS

Internship subject: Robotic task scheduling for industrial applications

CONFIDENTIALITY QUESTIONNAIRE

The internship will include the writing of a thesis and its defence before a board of examiners. Some confidential information on your Company might be included in it.

- Please let us know of your requirements regarding the circulation of the thesis and the organisation of the oral presentation. In the absence of a reply from your company, the thesis will be considered non confidential and the oral presentation will be public.
- We would like to draw your attention to the fact that a thesis is a precious source of information on companies and employment opportunities for our students. It is also a mean to enhance communication on your company towards future students.
- A confidential thesis is referenced but not accessible despite its technical and scientific value. We encourage our students to include confidential information in an annex to enable access to the thesis without disseminating confidential data.

1- Thesis Confidentiality

☒ This thesis is not confidential

☐ This thesis is confidential for a period of 5 years after the oral presentation

☐ This thesis is confidential indefinitely

2- Oral Presentation

☒ All professionals interested by the subject matter can attend

☐ Only members of the jury and participants vetted by the company can attend

For the company,
Date : 19/07/2021

The student,
Date : 19/07/2021

Update : 21st OCT 2016