uc3m | Universidad Carlos III de Madrid

Bachelor in Computer Science and Engineering

2020-2021

Bachelor Thesis

"Scrum project scheduler based on search algorithms"

Rodrigo Escribano Cifuentes

Tutorship

Carlos Linares López

Leganés, 23rd June 2021

SUMMARY

Nowadays it is not uncommon for software projects to fall under troublesome development periods that may turn the whole endeavour into a failure. Whether it is because of a lack of proper management or an inadequate estimation of resources, most of the problems could have been resolved with a proper planification phase before the beginning of the project.

The aim of this work is to provide a software tool that, with an input of employees and tasks, can generate an schedule in advance. This schedule is to serve as an early estimation of the project duration as well as a roadmap for all the people involved.

The software will be developed as a Constraint Satisfaction Problem (CSP) and solved using search algorithms. All these techniques are first explained in the document, followed by a detailed explanation of how they were implemented and the results. An study of other factors such as the project budget and the socio-economic impact and regulation is also included.

Keywords: scheduling, constraint satisfaction problem, search algorithm

DEDICATION

This work is dedicated to all of the people who have made this 6 years of engineering possible to complete believing in me throughout all this time, and to those that always knew that I would en up doing this degree and pushed me towards it.

To my parents, for encouraging me to pursue this career and not give up, all while not pressuring me and offering all the help they possibly could.

To Vicente Brox, for helping me getting out of the lowest point of my career and grow up as and adult pursuing my ambitions and dreams.

To all my close friends who have listened patiently to my rants while giving me support and entertainment during these arduous years of work.

To Carlos Linares, the tutor of this work that never doubted of its completion and pushed me towards realizing the whole scope of my ideas.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Project motivation

In the software development area it is not uncommon for projects to get delayed, postponed, or even canceled, and whilst sometimes these can be a cause of a lack of time or liability, this work will be centered in those that fail because of bad management and planning. Ultimately, all the problems mentioned before could have been avoided if a proper estimation of resources had been made. In order to make a proper estimation, the correct tools are needed, as well as some time dedicated to these affairs before the project is started.

There are plenty of project management tools in the market that offer compatibility with Kanban boards [1] or the Scrum framework [2], but their offerings vary greatly. Free ones leave all the work to the end user and simply connect the tasks and represent them with the given information, and other truly professional ones have to be purchased and do not necessarily offer an automatically generated planning. This is why the end product of this work is to design and implement an scheduler for projects that use the Scrum framework.

By automating the generation of a schedule, a prediction can be made of whether there will be enough time to complete the tasks as well as an overall view of the project from the very beginning. Being able to launch the software again at any time in the development is also crucial, as changes that affect the schedule will inevitably surface and the team will have to adapt to those, informing the client or team leader as well.

An extensive planification of a project also allows for a tightened estimation of the tasks and thus, a more realistic cost forecast. It also makes for a more realistic presentation towards the client that may have to pay for the project realization and will end with a more accurate view of the whole timing and cost of the endeavour.

Apart from real business applications of the software, the implementation of the algorithms that make it possible will imply a deeper understanding of search algorithms and the chosen programming language, as well as an study of parallelism and concurrency applications in the environment. All of these are topics that are becoming more and more used in modern applications as the hardware improvements introduce a greater number of cores in processors with expanded capabilities.

Lastly, focusing the software towards complying with Agile Development, specially Scrum, ensures it is compatible with modern developments that are every day leaning more towards these methodologies. But as these are not by themselves the only requirement for a good software planning, the project scheduler will prove to be an additional useful tool.

Essentially, many of the issues that arise in software development can be solved even before the project has started, which is the objective of this scheduler. Furthermore, creating

the software from scratch will allow a better implementation of the discussed problem that is optimized using modern standards and techniques.

1.2. Document structure

This document has been structured in the following manner:

- **Chapter 2: State of the art**

  An introduction to the essential concepts to be discussed throughout the document such as search algorithms and Constraint Satisfaction Problems.

- **Chapter 3: Objectives**

  What is to be accomplished with the developed software, its requirements.

- **Chapter 4: Development**

  A detailed explanation of every part of the developed software divided in three parts: Analysis, Design, and Implementation.

- **Chapter 5: Results**

  A battery of tests done with the finished software to analyze its efficiency, results, and how it behaves under different inputs.

- **Chapter 6: Conclusions**

  A look at what was accomplished with the software from a results and a personal growth standpoint with a reflection upon the hardest parts of the process.

- **Chapter 7: Project planning and budget**

  The project planning as seen in a Gantt chart and the hypothetical budget it would require if it were to be developed professionally.

- **Chapter 8: Regulation and Socio-Economic Impact**

  A look into the current regulation that may affect the software as well as the impact it may have in the current economic environment with similar programs.

- **Chapter 9: Future Work**

  Several ideas for the future that can be applied to the software to improve it or add more characteristics.

- **Appendix A: Solution Sample**

  A sample of what a solution looks like, including the input and output files.

2. STATE OF THE ART (METHODOLOGY)

This chapter serves as an introduction to the methodologies and algorithms to be discussed in the document, them being mainly the Scrum framework, the modeling of the software as a Constraint Satisfaction Problem, and the use of search algorithms for its resolution.

2.1. Scrum framework

Scrum is a framework for the development of projects created mainly by Ken Schwaber and Jeff Sutherland throughout the 1990s [3] and later fully described with Mike Beedle in 2002 [2]. It is meant to help teams of software development to be more productive and adaptive to changes with its application helping achieve a set of goals effectively and without delays [4].

There are many terms defined in Scrum but the main ones treated in this document are the following:

- **Sprints**

  When Scrum is applied, the development is divided into Sprints, which are periods of less than one month with specific goals in mind. During these periods Daily Scrums happen and the backlog stated for said Sprint is fulfilled. At the end of the period a meeting with the client is expected, and the goals achieved in the Sprint are reviewed as well as the backlog for the next Sprint.

  Having the project split into specific time frames allows the team to focus on specific tasks at a time and have a more organized development. The periodic meetings with the client help identify problems or disagreements in the product early as well as provide small deliveries that keep the client happy.

- **Backlog**

  The backlog term is used to describe the tasks that are still left to be fulfilled. An initial backlog is performed before the start of the development and is meant to be ordered and revised periodically, adding or removing tasks as the project progresses.

- **Daily Scrum**

  One of the most essential concepts in Scrum is the application of a daily meeting called Daily Scrum. This gathering is meant to be not a moment for development discussion or doubt solving, but rather a small insight into what each member of the team is going to be performing and what has been done since the last Daily. This is also the moment when the backlog is revised and the status of the Sprint is analyzed.

2.2. CSP: Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) is defined as a mathematical problem where a set of variables must satisfy a set of constraints [5] [6]. This kind of problems can have total or partial solutions, depending on if all objects must be instantiated with a value or only some of them, respectively.

In a CSP, the variables, domain and constraints are usually modeled using a tuple as such: R = (X, D, C) where:

- X are the variables
- D is the domain of the variables
- C are the constraints the variables must meet

Using search algorithms it is possible to iterate over the variables and its domains to find solutions that satisfy the constraints. There are a wide variety of algorithms that can be used so solve CSPs and each of them adapts better to different situations, meaning it is important to explore as many as possible. The problem described in this paper is to be modeled as a CSP and to be solved with the use of different search algorithms described next.

2.3. Uninformed Search algorithms

Search algorithms are procedures which solve search problems by retrieving data from a certain data structure represented as a graph. This section will cover the different algorithms and whether they can or cannot be applied to the problem given their characteristics [7].

When choosing an algorithm the completion of the following characteristics wants to be verified:

- Optimality: The algorithm guarantees that the best solution will be found.
- Completeness: The algorithm guarantees that all the solutions will be found, if more than one.

2.3.1. Breadth-First Search (BFS)

In Breadth-First Search, a graph is traversed by levels, meaning that each width level is completely traversed before advancing to the next level, essentially, traveling left to right and up to down. Whilst this method guarantees that all nodes will be traversed and thus

that a solution will be found (complete), it carries a heavy cost in computation memory that grows exponentially with the size of the graph.

An application of the algorithm can be observed in the following graph:

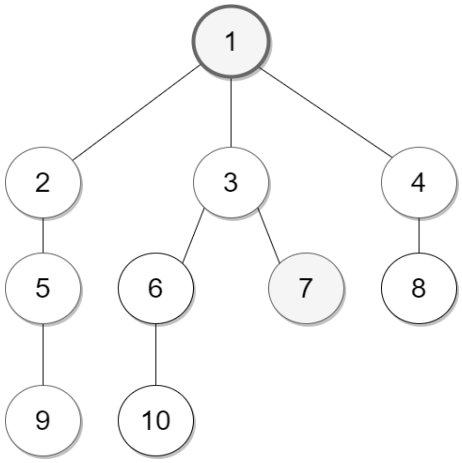


Fig. 2.1. BFS diagram

2.3.2. Depth-First Search (DFS)

In Depth-First Search, a graph is traversed in its depth, meaning that starting from the first node all of its children are visited until the maximum depth is reached before proceeding to the next node of the first branch. Essentially, traveling up to down from left to right. Once the furthest level is reached, backtracking is required to go back to the other unexplored nodes until the whole graph is traversed. This backtracking is required to guarantee the completeness of the method, as otherwise only one path is travelled not being necessarily a solution. Prioritizing depth and thus finding solutions rather than traversing the whole graph helps reduce the memory consumption of the algorithm and accelerate the process in problems where there are few solutions or the memory consumption grows exponentially.

An application of the algorithm can be observed in the following graph:

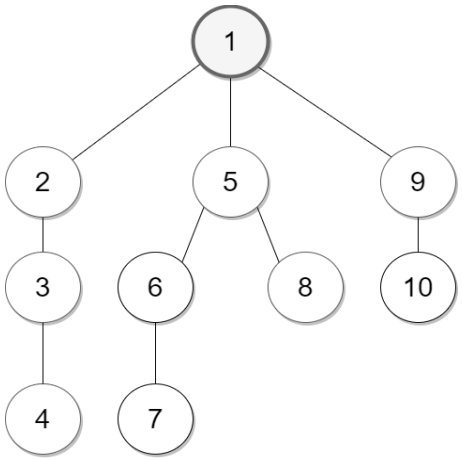


Fig. 2.2. DFS diagram

2.4. Informed Search algorithms

More advanced algorithms exist that require some form of additional information that helps the algorithm decide which path to follow when traversing the graph. This information is taken as a heuristic function, which estimates the cost from the current node to the goal state [8]. This function does not guarantee the best solution, but rather a favorable one helping reduce the time of the overall algorithm execution.

When choosing a heuristic function admissibility is desirable. This implies that the heuristic never overestimates the cost of reaching the goal, which is achieved when the estimated cost is always lower or equal than the actual cost of reaching the goal state.

2.4.1. A*

The A* (A-star) algorithm, first described in 1984 [9], traverses the graph in a similar fashion to the BFS algorithm but improves its performance by applying a heuristic. To select the next node to expand, an estimated cost to the objective node is calculated by minimizing the following formula:

$$f(n) = g(n) + h(n); \text{ Where:}$$

n is the current node

g(n) = the cost from the starting node to n

h(n) = heuristic estimation of the cost from node n to an objective node

If the heuristic is admissible, A* guarantees a solution with the least-cost path, thus being optimal. Regardless, the algorithm is guaranteed to be complete if the graph is finite and there are no negative weight edges. Given that all generated nodes are stored in

memory, it is very demanding in regards to memory, but it is still considered one of the best algorithms if the right heuristic is applied.

An application of the algorithm can be observed in the following graph:

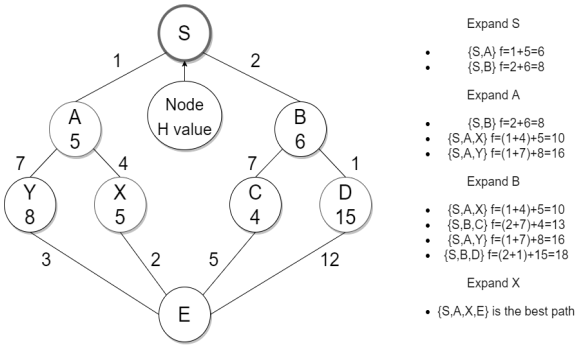


Fig. 2.3. A* diagram [10]

2.4.2. Branch and Bound (B&B)

The Branch and Bound algorithm, first described in 1960 [11], explores branches of a tree generating subsets of a complete solution. Once the solution(s) of a branch is found, it is compared against the upper and lower estimated bounds on the optimal solution, and is discarded if it cannot provide a better approach [12].

Unexplored nodes in the tree generate children by partitioning the solution space into smaller regions that can be solved recursively (branching), and rules are used to prune off regions of the search space that are probably suboptimal (bounding). Once the entire tree has been explored, the best solution found in the search is returned.

In order to estimate these bounds, a heuristic is necessary, depending the efficiency of the implementation on said estimation. If there are no bounds, the algorithm becomes a regular exhaustive search.

An application of the algorithm can be observed in the following picture where the objective is to find a subset of 1, 2, 3, 4, 5 that adds to 10:

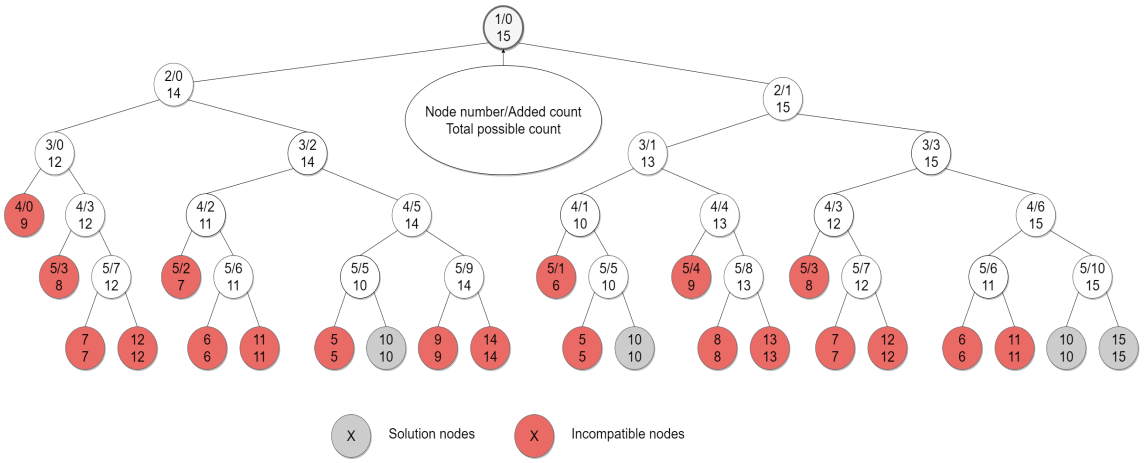


Fig. 2.4. Branch and Bound diagram [13]

2.4.3. Beam Search

The Beam Search algorithm, first named as such in 1977 [14], explores the nodes of a level expanding the most promising ones according to a parameter call beam width (K). In each level, the nodes are sorted by a selected heuristic function and the K best are expanded, repeating the process at each level until reaching a maximum of K solutions at the last one, if any.

Once again, the quality of the solution depends on the selected heuristic, but also on the width of the beam. The higher the beam, the more nodes that are expanded and the higher the chance is to find a better solution. If the width is set to infinite, the algorithm becomes Breath First Search, but otherwise it is not optimal as the best solution is not necessarily guaranteed.

An application of the algorithm can be observed in the following graph:

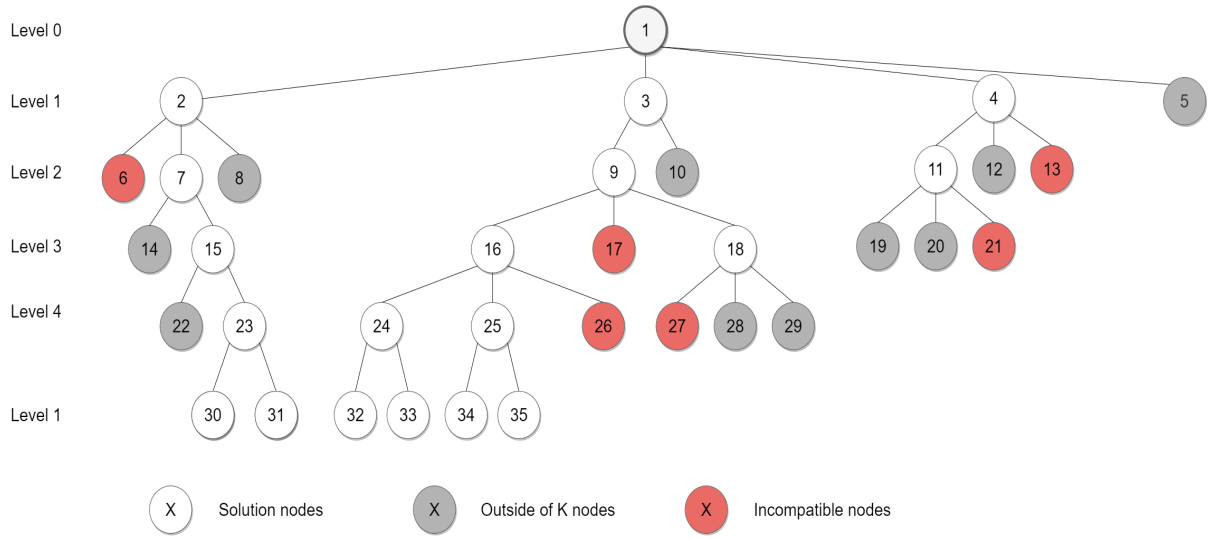


Fig. 2.5. Beam Search diagram with K=3 [15]

2.5. Conclusions

With all the main aspects of the environment that surrounds the project revised, it is time to select the appropriate search algorithms to implement and describe the use of the Scrum framework.

Given that the problem to be solved has a fixed depth length that is the number of tasks to be completed in a project, whilst growing exponentially in its amplitude with the number of employees and the time to complete said tasks, search algorithms based on depth are to be used. The selected algorithms are Depth-First Search and Beam Search as they can be adapted to the tree-like structure of the problem and offer fairly simple implementations.

Regarding the Scrum framework, the software will allow the schedule to be calculated in a specific time frame of weeks that helps adequate the result to the necessities of the project in terms of Sprints. Its full backlog and team will be loaded as input and Daily Scrums will be contemplated when performing all the calculations and algorithms such that the final timetable is correct and fits all the framework characteristics.

3. OBJECTIVES

The main objective of the work is, given some input data, to develop a software that is able to output an optimized schedule for a project that follows the Scrum framework. These input data will be the information of the people involved in the project (name and role) and the details of all the tasks that must be completed (name, duration, and compatible roles).

The idea is for the software to be run at the beginning of each Scrum Sprint with the data available at that time, and then to be run for each subsequent Sprint. Setting bigger time frames should also be possible in case the Scrum framework is not being followed. A correct scheduling is key in software development and avoiding it can lead to delays, crunch time, and imperfections.

Regarding the technicalities of the project, the software will be developed under the latest C++ 17 standard [16] and following the Scrum framework as found in the official documents [4]. The inputs will be read as CSV files and this document written using LaTeX[17].

Specific objectives are as follows:

- **Data input**

  The software must receive the input as two different files, one for the tasks and one for the employees involved in the project. Both files must have a clearly defined structure and be designed to optimize the parsing of the data, avoiding unnecessary duplication.

- **Data parsing**

  The input data must be correctly verified and parsed into the proper data structures to ensure the integrity of the information and avoid any bottlenecks when reading such data. The gathered information will be treated as invariant and must be available at all times.

- **Configuration and parameters**

  As the scheduler will take into account all the Scrum framework, the variability of it must be taken into account when launching the software. Thus, the user must be able to set the duration of the Sprint in weeks, the schedule of the project, as well as the initial date of the project.

- **Scrum involvement**

  The Scrum framework can be followed in several ways and not necessarily in its full capacity. In this software, the following characteristics will be taken into account:

  - Sprints
  - Daily Scrums
  - Backlog
  - Scrum roles

- **Results output**

  A Scrum board is typically used in software projects that follow this framework, in which the current status of the Sprint is displayed, showing the remaining backlog. To emulate this kind of board, the software must output each Sprint planning in an organized schedule easily readable and exported to other platforms.

- **Search algorithms**

  To ensure the highest performance of the algorithm, at least two different search algorithms will be implemented, mainly Depth-First Search and Beam Search. Both alternatives must be analyzed and tested thoroughly.

- **Testing**

  All elements of the software must be fully tested to ensure no errors are found and the output is feasible and optimized under all input conditions. The speed of the software as well as its reliability, viability, and performance under heavy workloads will decide whether the system is suitable for real world usage or not.

- **Optimization**

  Where a bottleneck is found, optimizations must be ensured via parallelism or multi-threading to speed up the execution of the software. These additions must, in no way, affect the final result of the software by altering the data or losing information.

4. DEVELOPMENT

4.1. Analysis

This chapter aims to make an analysis of the problem and determine the requirements the software will have to meet. It will begin by briefly summarizing the current state of affairs, and follow by stating the requirements and some use cases.

4.1.1. Current state of affairs

In these days, and specially after the impact of Covid-19, pretty much every business based on software development uses a form of project management.

Whilst there are plenty of tools used everyday in the software business to manage projects, whether they use the Scrum framework or not, none of them offer an automatic scheduler based on an input, but rather a simple management tool for the backlog.

This presents the need for the project leader, or the Scrum master, to manually manage the order and importance of the tasks to be completed, which can lead to a mismanagement of resources, time, or complexity. By automatizing this task, a whole view of the project and the time it may consume can be seen, which helps into the organization of other tasks of the project, as well as to give a better estimation to the client.

When developing software, anything can change at any time, from a sudden error in code to the sick leave of an essential programmer, which results in a planning that is very volatile and never final. This means that the project scheduler to be done in this work must be able to accommodate to those changes and recalculate the schedule based on the most recent events.

Agile methodology is used in part to accommodate to these changes, as having daily meetings can help anticipate problems to come. As these methodologies become more common, it is ever so important to take them into account when developing new software, in this case the project scheduler. As Scrum is very well documented and established, it was chosen as a reference for agile methodology based projects.

4.1.2. Software requirements

These section will cover all the software requirements this work must meet, them being functional or non functional requirements. For each one of them and ID will be assigned, as well as a name, description, and test methodology.

Table 4.1. REQUIREMENTS TEMPLATE

ID	FR | NFR
Name	Identifiable name
Description	Brief but concise description
Test case	Steps to verify the requirement

4.1.2.1 Functional requirements

Table 4.2. FR01

ID	FR01
Name	Project scheduler
Description	The software must calculate an schedule of a project based on input of the tasks and employees
Test case	The solution is correct and applicable

Table 4.3. FR02

ID	FR02
Name	Scrum framework
Description	The software must take into account the Scrum framework
Test case	The solution correctly implements all the characteristics of the Scrum framework

Table 4.4. FR03

ID	FR03
Name	Solution output
Description	The software must output the solution to a file
Test case	The solution is correctly exported to the file and is readable

Table 4.5. FR04

ID	FR04
Name	Search algorithms
Description	The software must use search algorithms to find the optimal solution for the schedule
Test case	The solution correctly implements the search algorithms

Table 4.6. FR05

ID	FR05
Name	Optimization and concurrency
Description	The software must be optimized and implement parallelism
Test case	The solution correctly implements parallelism and runs at its most optimized version

4.1.2.2 Non-functional requirements

Table 4.7. NFR01

ID	NFR01
Name	Programming language
Description	The software must be implemented using the C++17 standard of the C++ programming language
Test case	There are no warnings or errors in the code execution

Table 4.8. NFR02

ID	NFR02
Name	Input format
Description	The input for the tasks and employees must be in CSV format
Test case	The software correctly reads the files with no information loss

Table 4.9. NFR03

ID	NFR03
Name	Input parsing
Description	The input will be parsed to extract the relevant data and store it in local memory
Test case	The software correctly stores the files with no information loss

Table 4.10. NFR04

ID	NFR04
Name	Output format
Description	The output for the scheduled solution must be in CSV format
Test case	The solution can be correctly read with no information loss

Table 4.11. NFR05

ID	NFR05
Name	Scrum implementation
Description	The software must take into account the Scrum framework for its scheduling solution
Test case	The software correctly implements the Scrum framework

Table 4.12. NFR06

ID	NFR06
Name	Chosen search algorithms
Description	The software must, at least, implement the Depth-First Search and Beam Search algorithms to find its solution
Test case	Both algorithms are correctly implemented and deliver a proper solution

Table 4.13. NFR07

ID	NFR07
Name	Parallelism implementation
Description	The software's parallelism implementation must be done with accordance to the C++17 standard [16]
Test case	The libraries are used correctly with no warnings or errors

Table 4.14. NFR08

ID	NFR08
Name	Operative system compatibility
Description	The software must function properly under the Windows operating system
Test case	The software is executed in Windows with the same result as inside the programming IDE

4.1.3. Use cases

This section will represent how the software is intended to be utilized with a Use Case diagram. This diagram shows the agents that interact with the system and how the system performs internally to those inputs. In this case there is only one user interacting with the software whilst there are two inputs (tasks and employees) and one output.

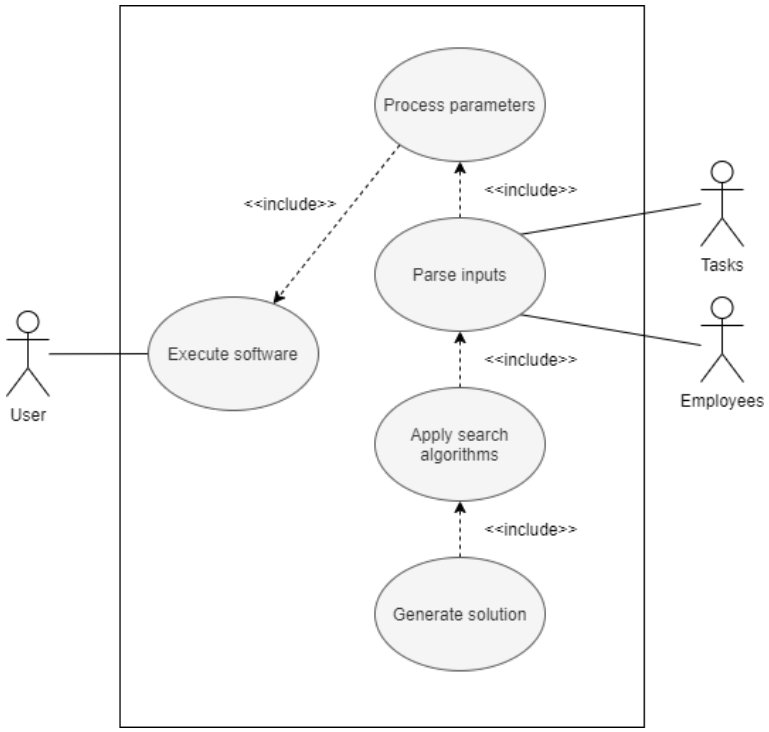


Fig. 4.1. Use case diagram

Table 4.15. USE CASE TEMPLATE

Use case	Name of the use case
Actor	Name of the actor
Description	Brief but concise description

Table 4.16. USE CASE 1: EXECUTE SOFTWARE

Use case	Execute software
Actor	User
Description	The user executes the software with the desired parameters and awaits the end of the processing

Table 4.17. USE CASE 2: PROCESS PARAMETERS

Use case	Process parameters
Actor	-
Description	The software processes all the input parameters and changes the execution accordingly

Table 4.18. USE CASE 3: PARSE INPUTS

Use case	Parse inputs
Actor	Tasks and employees
Description	The software parses all the input data of the Tasks and Employees actors storing it in local memory for further use

Table 4.19. USE CASE 4: APPLY SEARCH ALGORITHMS

Use case	Apply search algorithms
Actor	-
Description	The software applies the corresponding search algorithms to find the solution(s)

Table 4.20. USE CASE 5: GENERATE SOLUTION

Use case	Generate solution
Actor	-
Description	The software outputs the solution(s) in CSV format

4.2. Design

This section will cover the design of the software visualized in diagrams and detailed explanations of the algorithms and processes used.

4.2.1. Class Diagram

The class diagram should help getting an overall view of the distribution of the code as well as the different tasks that are performed within it.

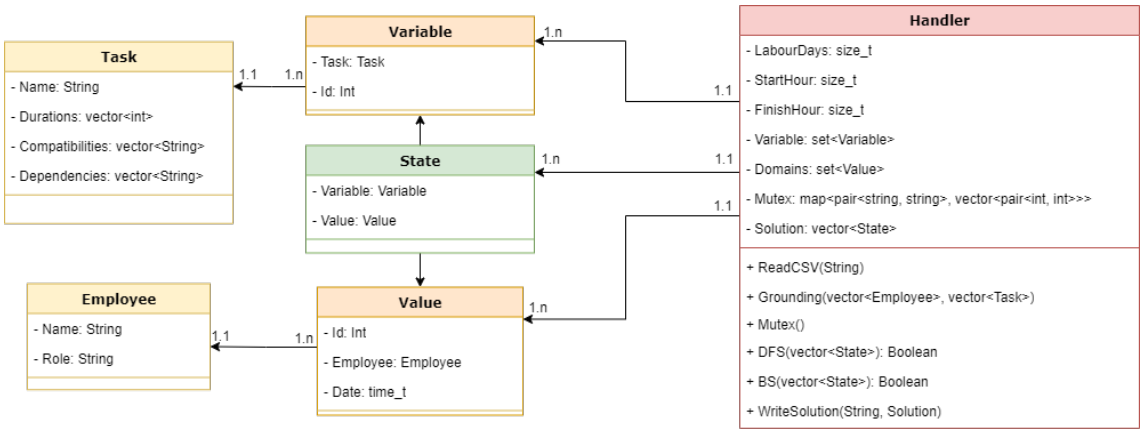


Fig. 4.2. Class diagram

Each one of the classes is described as follows:

Table 4.21. CLASS TEMPLATE

Name	Name of the class
Dependencies	Dependencies of the class
Description	Brief but concise description

Table 4.22. EMPLOYEE CLASS

Name	Employee
Dependencies	None
Description	All of the employees of the project to be assigned (or not) to tasks. They have a role that can vary between employees.

Table 4.23. TASK CLASS

Name	Task
Dependencies	None
Description	Each software project has several tasks that must be completed and distributed among the employees. Each one of these tasks has a duration in estimated hours as well as certain employee roles that can perform the task and dependencies on other tasks.

Table 4.24. VARIABLE CLASS

Name	Variable
Dependencies	Task
Description	A Variable is the combination of a Task and an identifier.

Table 4.25. VALUE CLASS

Name	Value
Dependencies	Employee
Description	Each Variable can take different Values. These Values form the domain and are composed of an identifier, an Employee, and the day and time (date) at which the Task is performed.

Table 4.26. STATE CLASS

Name	State
Dependencies	Variable and Value
Description	A State is composed of a Variable and its assigned value. In order, a collection of states represents the schedule for the project.

Table 4.27. HANDLER CLASS

Name	Handler
Dependencies	Variable, Value and State
Description	The Handler acts as the main class that communicates with all the others, reads the input, executes the search algorithms, introduces Scrum framework, and processes the solutions.

4.3. Implementation

The detailed explanation of the implementation of the search algorithm and all the necessary preparations that have to be performed before it are specified now.

4.3.1. Grounding and mutex

In a Constraint Satisfaction Problem, the variables, domain and constraints are usually modeled using a tuple as such: R = (X, D, C) where:

- X are the variables
- D is the domain of the variables
- C are the constraints the variables must meet

For this particular case, each Variable is a Task that must be fulfilled. Each Task can be done by a specific combination of Employee, day, and time (hour), and all of those combinations comprise the Domain of the variable. Once a Variable has been assigned a specific Value of its Domain, several other Values (of that and other Variables), become unavailable or prohibited. This ensemble of prohibited Values is stored in a list of pairs called mutex list.

In this software implementation, Values are calculated independently of the Domain. This way there is a set of unique Variables and unique Values and each time a combination of Variable and Value is going to be calculated, it is first checked whether that combination is possible.

During the grounding process, the software does all the preparation for the execution of the algorithm by performing two tasks: generating all the possible Variables, and calculating the domains of the Variables.

To generate the Domains, all the possible combinations are calculated by iterating Employees for each day of the week during the introduced period of the project. It is at this moment that the unfeasible combinations and the combinations at the Scrum time are discarded to save iterations in the process.

- **Unfeasible combinations**

  The main unfeasible combination to be discarded is the Employees being assigned outside of the schedule.

Apart from these calculations, there are the so called mutually exclusive *mutex* lists, which indicate those situations that become impossible once a Variable has been instantiated. The use of these lists helps alleviate the load of the algorithm as a lot as the combinations are avoided by checking the lists instead of being generated and then discarded. The mutex process is performed right after the grounding [1] [2] [3].

The nomenclature to be used in the definition of the mutex is the following:

[1] R. D. Olivaw. (2017). "Ai-csp: definition, constraint propagation, backtracking search, local search, problem structure." [18]

[2] V. K. Chaudhri. (2011). "Constraint satisfaction problems" [19]

[3] H. Ghaderi. (2011). "Backtracking search (csps)" [20]

- T for Tasks
- E for Employees
- D for each day of the week
- H for each working hour of a labour day

There are three types of mutex in the software:

- **Task mutex**

  By design, once a Task has been instantiated, all the other Values that Task may take become invalid. This is the easiest kind of mutex and allows us to easily free many combinations. It is implicit in the design.

  - Mutex1: For each Task, the number of mutex combinations it has is each Employee that can perform that Task, times the days of week, times the hours of each day, minus one Value that is the one that has been assigned.

$$\forall \text{Task}, \text{Mutex}_{\text{Task}} = E \cdot D \cdot H - 1$$

- **Employee mutex**

  Similarly, one Employee cannot be assigned to two Tasks that overlap. This means that for the duration of the Task, all the other Values that include the Employee can be safely discarded.

  - Mutex2: For the duration of the task, the Employee assigned to that Task cannot perform any other task.

$$\text{Assigned Task being } \mathrm{T}_0 \text{ with Employee } \mathrm{E}_0$$
$$\forall Task \neq T_0, Mutex_{\text{Employee}} = E_0 \cdot D \cdot H$$
$$T_{\text{i.date}} < T_{0.\text{finishDate}}, T_{0.\text{date}} < T_{\text{i.finishDate}}$$

- **Dependency mutex**

  Tasks that depend on another Task cannot be assigned before the preceding Task has been performed. This effectively eliminates all Values of the Task at a time preceding the current one.

  - Mutex3: If a Task 1 depends on a Task 2, all Values of Task 1 that precede Task 2, and that overlap it, are eliminated.

$$\text{Assigned Task being } \mathrm{T}_0$$
$$\forall Task \neq T_0, \text{Mutex}_{\text{Dependency}} = E \cdot D \cdot H$$
$$T_{0.\text{Date}} < T_{\text{i.finishDate}},\ T_{0.\text{Name}} \in T_{\text{i.Dependencies}}$$

The following constant variables that are asked as input at the initialization of the software are also included:

- Daily schedule of the project.
- Time of the daily Scrum reunions, if any.
- Maximum or expected duration of the project in weeks.

4.3.2. Helper functions

In the software implementation there are a couple of algorithms that are repeatedly used and compose a big part of the logic. These algorithms have been iterated through and debugged several times as they are an essential part of the software.

4.3.2.1 Task finish time

A key part of the system is knowing when a Task ends. It is an essential part of the mutex calculation, the Beam Search heuristics (section 4.3.3.2) and the insertion of the Scrum dailies.

An error in the calculation leads to incorrect solutions that do not match with the duration of the Tasks and can produce conflicts with weekend days. Given the lack of proper time libraries and methods in C++, programming and debugging this characteristic has proven more difficult than in other programming languages.

C++ has always had the variable type *time_t* for representing time as the number of seconds elapsed since 00:00 hours, Jan 1, 1970 UTC, what is called epoch time. This allows for time to be easily stored and also manipulated by addition and subtraction of seconds, but representation in a regular date format requires additional work.

C++ also has the *tm struct*, which contains individual parameters for elements such as day of the month or hour of the day, which in combination with the epoch representation allow for the correct manipulation of time variables to calculate the end of a Task.

It is important to notice that *tm* does not work as other similar implementations in languages like Java, as adding time to the *struct* will not move time forward or back, it will just add it to the *struct* parameter. This means that for manipulating time it is necessary to use *time_t*, whilst for representing it or checking if it is weekend, *tm* is used. Thus, the time variable is changed between the two types each time an operation is performed.

With these notions clear, this algorithm is divided in three steps:

- If the Task spans more than a labour day, advance the time forward those many days.

- Add the remaining hours.
- If the time is past the end of the labour time, advance to the next day and add the extra hours.

For the last step, the lack of proper time functions becomes apparent. The difference in hours between the current time and the end of the labour day is calculated. If time is still in the same day, it is advanced until the next day at the start of the labour day, and if time is already at the next day, it is simply moved to the start of the day. It is at this point that the difference in hours is added and the final finish time is reached.

Just before the second step, if Scrum is enabled, an hour is added for each day that is passed, as it corresponds to the daily meeting. And in each step, if time is at Saturday or Sunday, it is advanced until Monday.

4.3.2.2 Domain and Mutex checking

Each time a Value is going to be tested or created, it is checked if it belongs to the Domain of the Task and/or if it is mutex with the already instantiated States.

To check if it belongs to the Domain of a corresponding Variable, the Employee of the Value is contrasted against the compatibilities of the Task. This is a very simple operation that does not carry a time penalty and is more efficient than manually storing the Domain of each Variable, both in time and storage costs.

To check if two Values of two different Variables are mutex between themselves, they are searched for in the mutex mapping which is composed of pairs of Variables mapped to pairs of Values. Using a Map allows for quick searching and reduces the time penalty in the software, but as will be seen in chapter 5, calculating the mutex is one of the two most complex and time consuming elements of the software.

4.3.2.3 Scrum dailies insertion

Once the search algorithms have produced solutions, the daily Scrum meetings must be inserted into them. As grounding does not produce possible Values at the Scrum dailies time, and every time a Task finishing time is calculated these meetings are taken into account, these Tasks can simply be created and inserted into the final solutions.

The Scrum dailies target all Employees and are performed at the designed time, thus, their parameters are static except for the day of the Task. A simple loop creates the Tasks and inserts them in the vectors of solutions advancing one day at a time and avoiding weekends.

4.3.2.4 Output formatting

Once all the data is obtained, it is transformed into an appropriate format that can be stored in a CSV file. This is based on creating a vector for each column of the file and writing each one of them to the file, writing first the name of the column. An additional "Finish" Task is added at the end of its column to mark the end of the schedule.

4.3.3. Search algorithms

For the final implementation of the software two search algorithms were selected: Depth-First Search and Beam Search. Given the characteristics of the problem them being:

- Tree-like structure (not a graph)
- High number of states
- Fixed depth of the tree
- High number of duplicate solutions

These two algorithms were selected as they offer fast search times that deliver good solutions and can be scalable in the case of Beam Search. The detailed implementation of both algorithms will now be described. First a high level description of each algorithm will be delivered followed by detailed explanations the key additional algorithms that had to be developed [4].

4.3.3.1 Depth-First Search

This algorithm prioritizes reaching the end of the tree choosing the first possible solution it can find. Thus, it is fast and simple to implement in a recursively manner.

The function receives a vector of non-initialized States that is passed as parameter in each subsequent call. In each call it is checked if the vector has all States instantiated, and if that is the case, true is returned to finish the execution of the algorithm in all depth levels.

If there are States to be assigned, the first one is selected and assigned the first Value of its Domain that is compatible with all the other Values that have already been assigned. If a Value is found, it is assigned to the State, pushed into the vector and a recursive call to the function is made. If there is no Value found, the instantiation of the current State is deleted and the algorithm backtracks to the previous assigned State to select another instantiation of that previous State.

[4] J. T. Point. (2018). "Informed search algorithms" [21]

In the end the algorithm returns a single solution it being the first one it has found. Thus, its feasible but most probably not very optimized or adequate. This is where Beam Search can provide better solutions by examining different possible Values for each State [5].

4.3.3.2 Beam Search

Beam Search is an iterative algorithm which for each State checks all the possible values selecting the K best ones according to a selected heuristic.

The quality of the algorithm is heavily dependant on said heuristic of which three kinds have been implemented:

- Heuristic 1: Earliest initial date of the last instantiated Task.
- Heuristic 2: Earliest finishing date of the last instantiated Task.
- Heuristic 3: Lowest value by calculating the sum of all the initial dates of the instantiated Tasks. (Added as epoch values)

These heuristics are all calculated in the same fashion: first the Values are sorted according to the metric, and then the first K Values of the vector are copied into a new one.

The function receives a vector of non-initialized States, a K number of Values that are kept in each step, the selected heuristic, and the number of S solutions that are desired. As K will be large, it is possible to select an elevated number of solutions. The vector of States is pushed into a vector of vectors which stores the K best solutions.

For each Variable, the whole vector of solutions is traversed, and for each solution all its possible Values for the Variable are calculated. These Values must be compatible with the already assigned Variables. Once all Values are computed, the K best ones are selected and pushed into new solutions. In this case, the best Values are the ones with the earliest initial date, which is essentially the first heuristic. Duplicated Values are also discarded in this step.

As this is done for each solution, there are now K^K possible solutions, which are purged using one of the metrics. Once all Variables have been assigned, the K best solutions the algorithm was able to find remain, from which the first S ones are selected according to the input value. For this the second metric is used as all instantiations are final and a this point what is desired is to finish the project at the earliest [6].

4.3.4. Multi-threading and parallelism

The key areas of the software, them being the mutex calculation and the Search Algorithms, take an increasingly amount of time that escalates with the complexity of the

[5] Reducible. (2020). “”Depth first search (dfs) explained: Algorithm, examples, and code” [22]

[6] J. Brownlee. (2018). ”How to implement a beam search decoder for natural language processing” [23]

problem. To reduce the time these algorithms take, different implementations of multi-threading and parallelism have been introduced [24].

The impact these optimizations make in the software is studied in chapter 5.

4.3.4.1 Mutex multi-threading

The mutex algorithm is essentially a group of nested for loops that perform some calculations and introduce data into the mutex map. The outermost loop is performed once per Variable, thus, it is possible to split the Variables into threads that perform the nested loops.

A traditional implementation of multi-threading would have the each thread block the map whenever data is going to be inserted, but this was not the selected implementation. Instead, one vector per thread is created and each thread outputs the calculations into those vectors. Once all threads are finished, the data is sequentially inserted into the mutex map in the main thread, without any data lost or corrupted.

The standard C++ thread library does not allow threads to return values, in this case vectors, for which the *std::future* library had to be used. This library uses *async* threads that return a *future* variable that eventually holds the result of the called function. This avoids the need to join the threads as the call to the *future* variable already handles that and only returns the result of the thread calculation once it has finished. Given that the selected PC for the testing has a 6 core processor, the implementation includes options for 1, 2, 4, or 6 threads.

4.3.4.2 Beam search sorting parallelism

A key part of the Beam Search algorithm is selecting the K best Values of each Variable. This is done by sorting the available Values from best to worst and selecting the first K ones. This sorting is increasingly demanding as the number of Values increase with the complexity of the problem, thus, accelerating its execution can deliver important performance gains.

C++17 introduced support for parallel algorithms into the standard libraries, which makes the implementation of such improvements to the sorting extremely easy. It is only necessary to indicate one of the three included policies in the sort call and the compiler does all the necessary work. The policies are the following:

- 'seq', which executes the sorting in the standard sequential fashion.
- 'par', which parallelizes the sorting.
- 'par_unseq', which parallelizes and vectorizes the sorting.

In this case vectorizing means that as well as using multiple threads, multiple calls to the function can be made per thread. For this use case, there is no need to perform multiple calls to the sort algorithm, thus both policies should perform similarly [7] [8] [9].

[7] B. O'Neal. (2018). "Using c++17 parallel algorithms for better performance" [25]
[8] R. Schulung. (2018). "C++core guidelines: Rules for concurrency and parallelism" [26]
[9] B. Filipek. (2017). "C++17 in details: Parallel algorithms" [27]

5. RESULTS

This chapter will cover the design and analysis of the tests applied to the final implementation of the software for the Scrum scheduler. As there are two main parts in the software, them being the treatment of the input Variables and afterwards the use of those Variables and Values with the search algorithms, this chapter will be divided in that same way.

All the tests were performed inside Visual Studio 2019 with a fresh boot of the operative system under these specifications:

- CPU: AMD Ryzen 3600X
- RAM: 2x8GB DDR4 @3600Mhz
- GPU: Nvidia RTX 3070
- Boot and installation SSD: Kingston A400 120GB
- OS: Windows 10 Pro 20H2 V.19042.928
- IDE: Microsoft Visual Studio Community 2019 Version 16.9.3
- Compiler: Visual C++ 2019 - 00435-60000-00000-AA228
- Programming language: C++ 17

The following tests are based on changing parameters of the variables to observe how they affect the performance and the results of the scheduler. As there are many possible configurations, the worst case scenario is observed to get the worst possible timings.

This is composed of several scenarios:

- Tasks that all have the same roles, meaning they can only be done by a single group of Employees.
- Employees that all have the same roles, meaning they can all do the same Tasks.
- Tasks that depend one over the other, meaning they have to be completed sequentially.

Thus, all tests are performed under these scenarios while changing the following parameters:

- Schedule: The schedule of the project

- Tasks (Variables): The Tasks the Employees must complete.
- Dependant Tasks (Variables): Tasks where each one depends on the previous one, creating a chain of Tasks that can only be completed one by one.
- Employees (Values): The Employees that will perform the Tasks.
- Weeks: Number of weeks the software schedules.
- K: Parameter of Beam Search that dictates the beam width, which is the number of selected nodes per Variable.
- BS algorithm: The Beam Search algorithm that dictates how the best K nodes are selected.
- Sort algorithm: The type of sort algorithm that is used in Beam Search,
- Threads: Number of threads in the mutex calculation.

To preserve the same environment across all tests, the number of points of information and static variables has been kept unchanged were possible. The following configuration was selected as the standard one, with unmentioned parameters being varied:

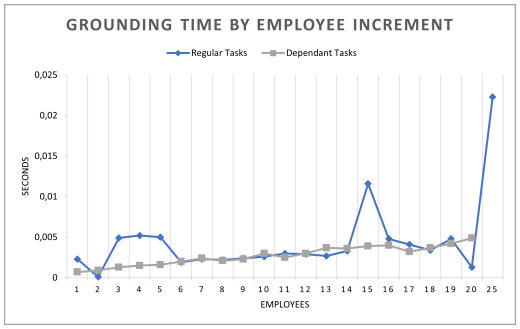
- Schedule: 9:00-17:00 Monday to Friday.
- Weeks: 2.
- K: 50.
- BS algorithm: Algorithm 2.
- Sort algorithm: par_unseq.
- Threads: 2.

5.1. Input preparation results

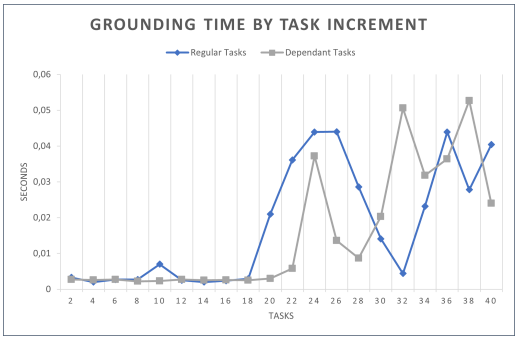
The input Tasks and Employees are parsed into objects and then treated inside the Grounding and mutex algorithms to establish the base parameters for the search algorithms.

5.1.1. Grounding

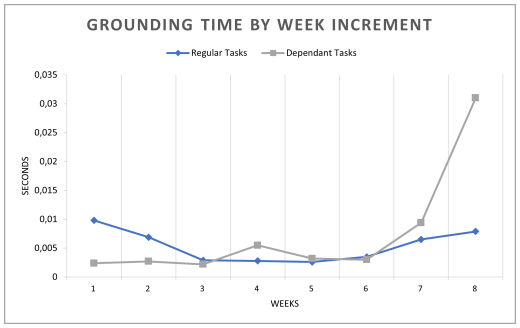
The grounding process simply parses the inputs into a set of Variables and a set of Values. As a set is composed of unique values, this process is very fast and its performance is dependant on the number of inputs and selected weeks.



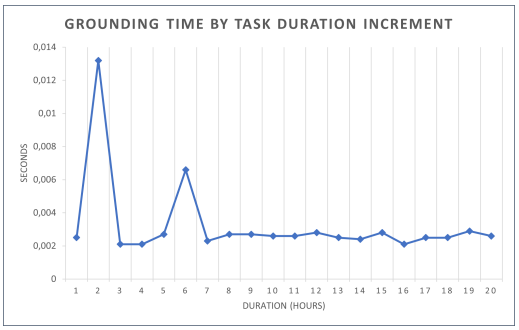
(a) Grounding by Employee increment



(b) Grounding by Task increment



(c) Grounding by week increment



(d) Grounding by Task duration increment

Fig. 5.1. Grounding time

As the graphs suggest, the grounding process is practically instantaneous no matter the number of input variables. It can be seen in graph (a) that there is a slight increment of time with each added Employee, specially uniform when Tasks are dependent. This cannot be extrapolated to the addition of Tasks or weeks as the variation within the times could be attributed to margin of error.

It does make sense that adding Employees increases the time slightly as its grounding process involves combining them with time frames, whilst the Tasks are only parsed into the corresponding set.

5.1.2. Mutex

The mutex process is heavily dependant on the compliance of the two types of mutex conditions, thus, if a Variable is not inherently mutex with other ones, its mutex calculation is fast. To observe how these conditions affect the timing, the worst case scenarios that maximize the number of mutex conditions have been tested. This is:

- Employees that can all do the same Task, meaning overlapping Tasks are maximized.
- Tasks that depend one over the other, meaning dependency is maximized.

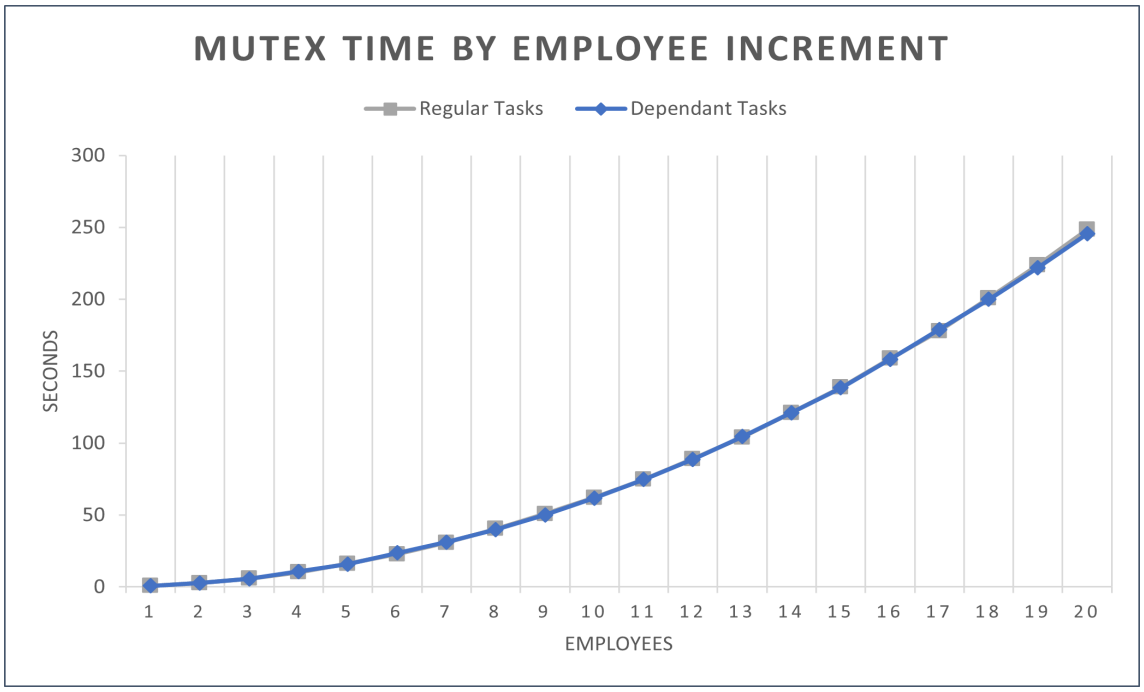


Fig. 5.2. Mutex time by Employees

The first test with an increasing number of Employees shows a polynomial increase of degree 2 in time but with little to no difference with the addition of dependant Tasks. This makes sense as dependency is inherently a characteristic of the Tasks and thus does not affect Employees.

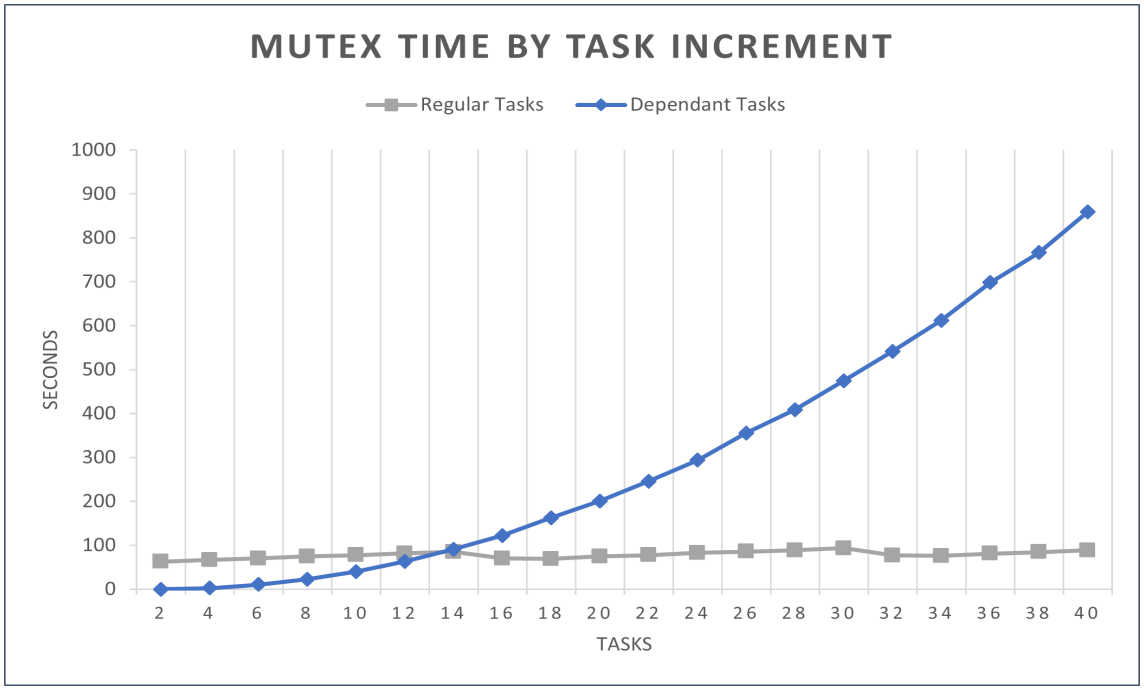


Fig. 5.3. Mutex time by Tasks

Contrary to what was seen in Figure 5.2, increasing the number of regular Tasks does not yield an increase in time as the only mutex it affects is the first type that is naturally avoided in the code.

However, if the Tasks are dependant on one another, the third type of mutex is affected, increasing the overall duration of the process polynomially again. It is important to notice how the scale changes with respect to the previous graph and now the worst case is taking upwards of 15 minutes.

Now the Task duration effect will be examined.

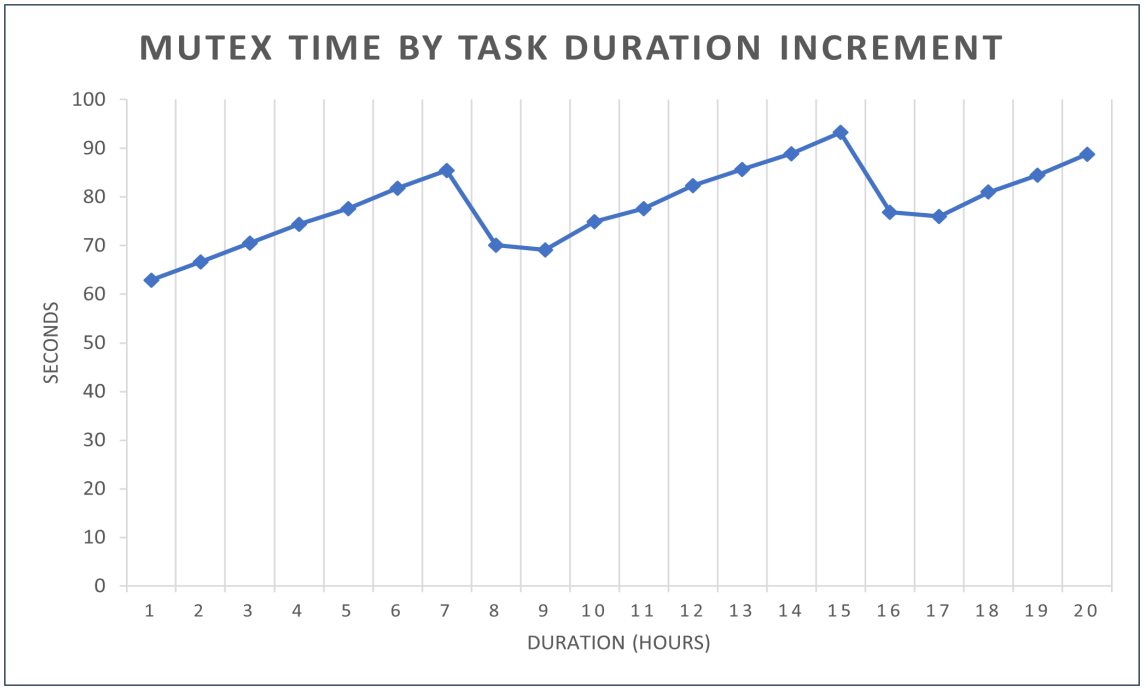


Fig. 5.4. Mutex time by Tasks duration

Interestingly, increasing the duration of the Tasks seem to follow a pattern where each 8 hours the time drops and then increases again. This might be due to days spanning 8 hours, meaning the Task duration calculation is the slowest when Tasks take those 8 hours or multiples of it, due to the way the algorithm is implemented. This said, the time is still overall increasing slightly with each 8 hours spanned.

It is also interesting to see how the total weeks of the project affect the mutex times, as increasing the days inherently creates more possible combinations in the grounding.

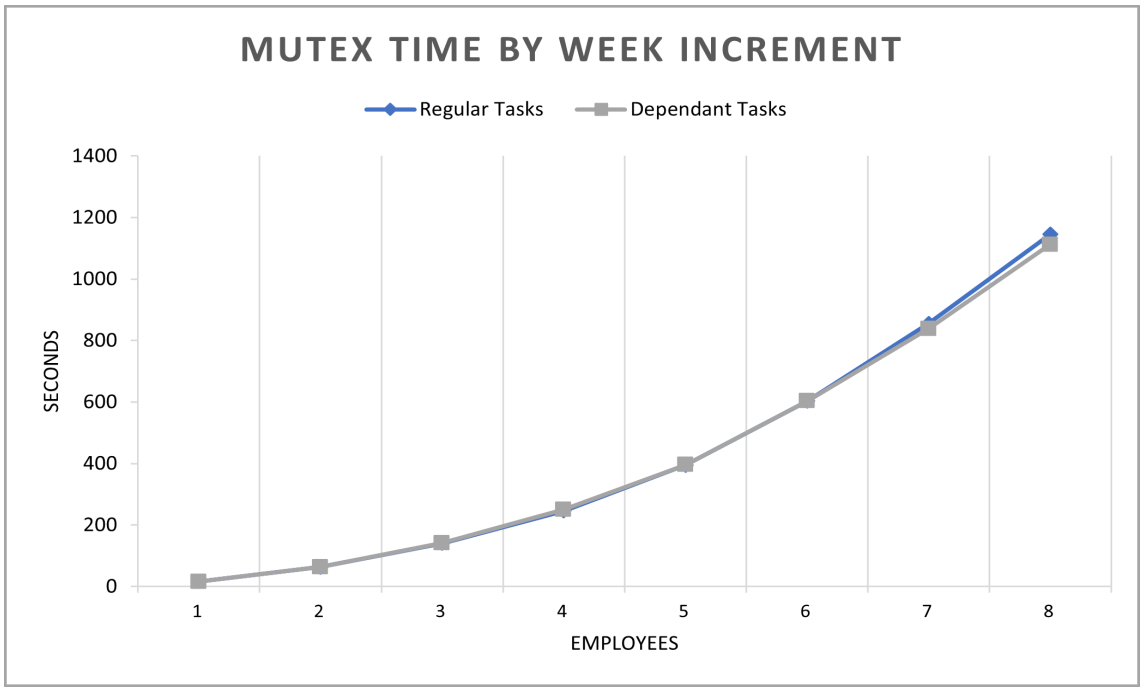


Fig. 5.5. Mutex time by weeks

A similar graph to Figure 5.2 can be observed, where growth is polynomial with the number of weeks but at a noticeably higher rate, ending with times of up to 19 minutes. More weeks essentially leads to more loop iterations in the calculation, which naturally increases time at a regular rate.

Being the corner cases examined, it is also important to observe how mutex behaves under normal conditions, where there is a healthy mix of Tasks performed by Employees of different roles.

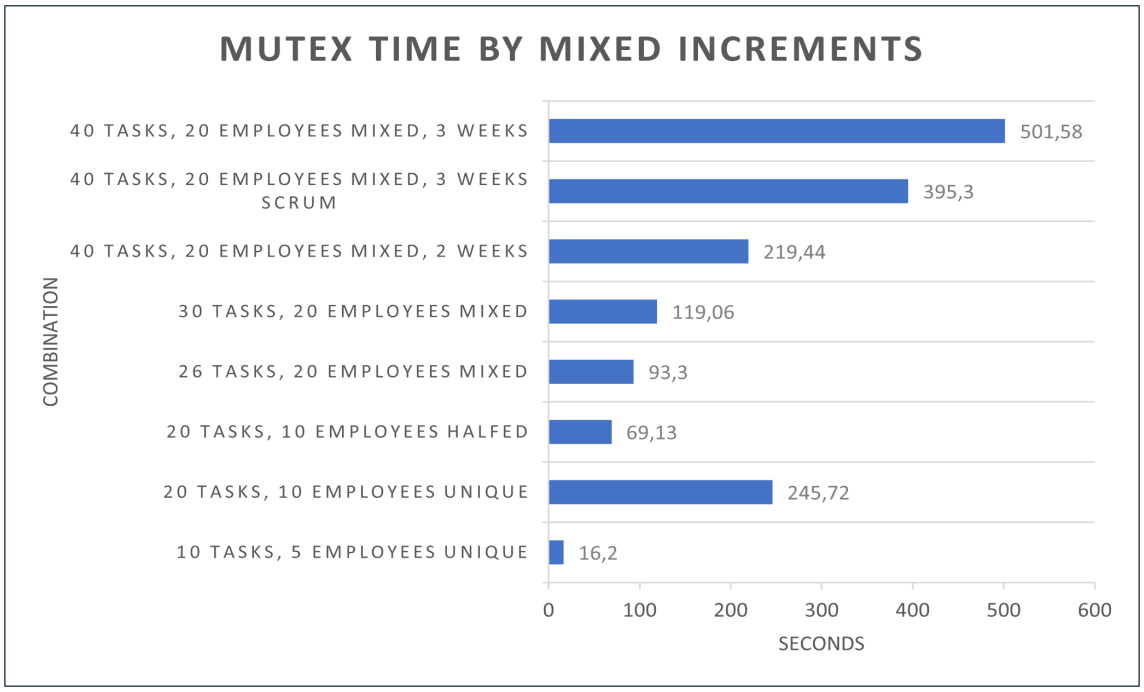


Fig. 5.6. Mutex time by mixed increments

As the graph shows, under a real case where 26 Tasks and 20 Employees are divided among 6 roles, the performance is much greater and the spent time reduced significantly even if the Variable and Values number is big. Increasing the Tasks to 30 and even 40 still yields times lower than the ones with Employees and Tasks with the same roles, it is only when increasing the weeks that time scales significantly.

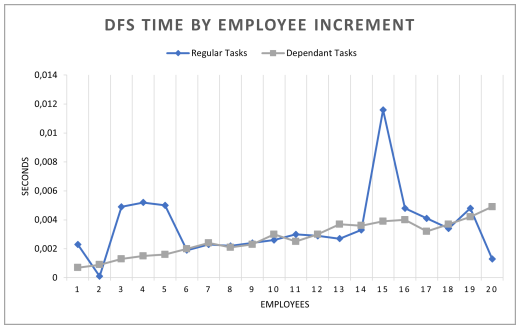
This is meant to show that real world performance is good, even though this whole chapter is focused on worst-case scenarios.

5.2. Search algorithms results

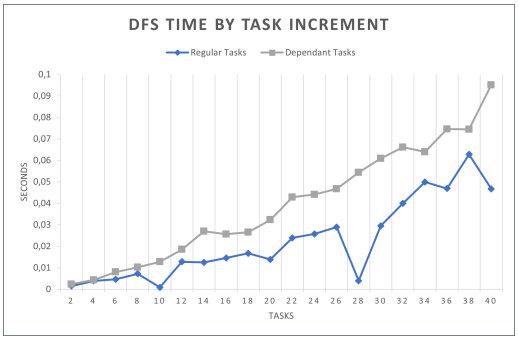
The second half of the software performs the search for a solution using the selected search algorithms. The performance will depend on the number of Tasks and Variables as well as the chosen algorithm and its own parameters.

5.2.1. Depth-First Search

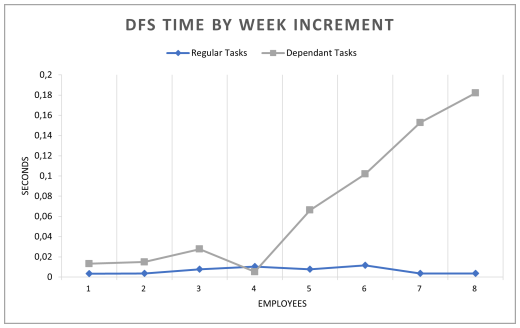
DFS is a fast search algorithm that only finds the first valid solution, thus its performance is great under all circumstances, but it is still interesting to see the time variation.



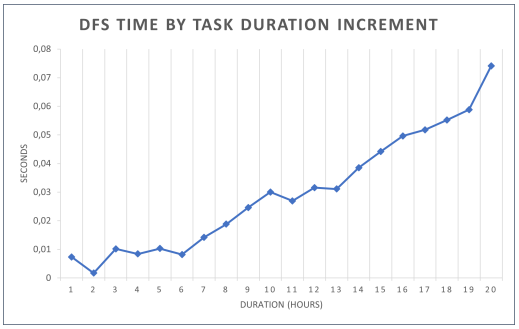
(a) DFS by Employee increment



(b) DFS by Task increment



(c) DFS by week increment



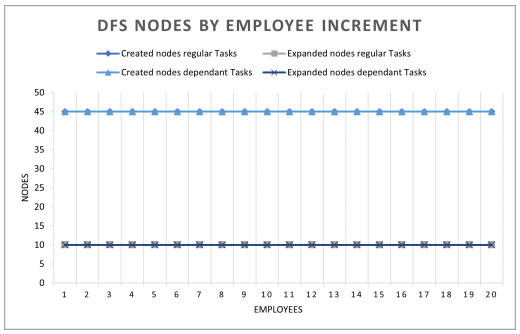
(d) DFS by Task duration increment

Fig. 5.7. DFS time

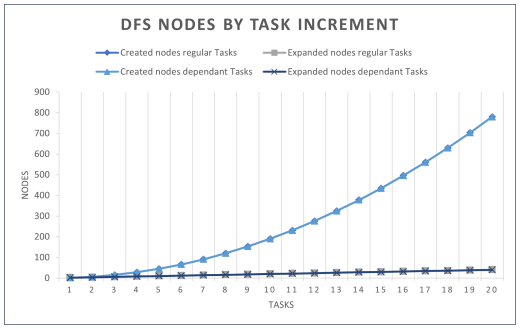
Once again, times are overall negligible, but in this case there is an appreciable increase with certain parameters. Employees do not seem to have a great effect on time but augmenting the Tasks leads to a linear increase that can also be seen from the 5th week if the Tasks are dependant.

All of these data makes sense as this implementation of DFS takes the first valid solution and this is not affected by the number of Employees, as they can all perform the same Tasks. Increasing Tasks however, enlarges the depth of the search three meaning DFS has to perform more iterations, increasing the total time of the algorithm. In the case of weeks, as can be seen in (c), the time increase starts to be noticeable from the 4th week.

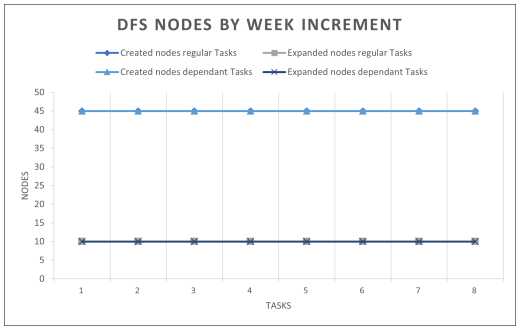
Now the number of expanded and generated nodes will be shown. If the created nodes are equal to the number of Variables, then the solution is found at the leftmost part of the tree and the algorithm is as fast as it can be. The more generated nodes that are expanded, the more time the algorithm takes and the harder the solution is to find.



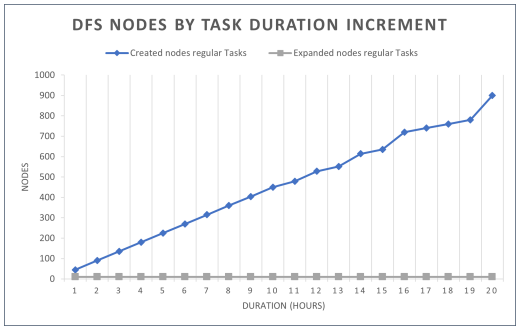
(a) DFS nodes by Employee increment



(b) DFS nodes by Task increment



(c) DFS nodes by week increment



(d) DFS nodes by Task duration increment

Fig. 5.8. DFS nodes

As DFS only takes the first plausible solution, under all cases the number of expanded nodes will be the number of Tasks that have been inserted as input. With this in mind, as the first solution is the selected one, increasing the number of Employees or weeks does not have an impact on the expanded nodes and thus on the result, that will be the same one in all cases. The number of created nodes does increase polynomially with the dependant Tasks as that imposes a new condition that must be checked for each node, increasing their number.

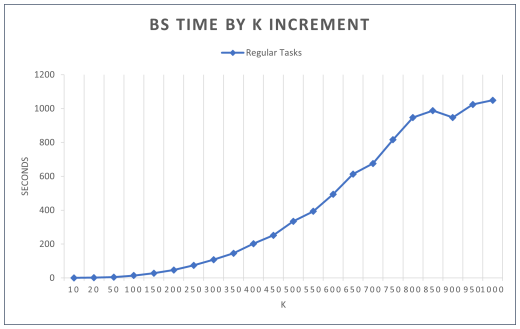
Interestingly, the duration of the Tasks does make an impact in the expanded nodes by increasing them in an strictly linear fashion. This makes sense as essentially the solution is being delayed by that increased number, augmenting the node value by the same amount each time.

5.2.2. Beam Search

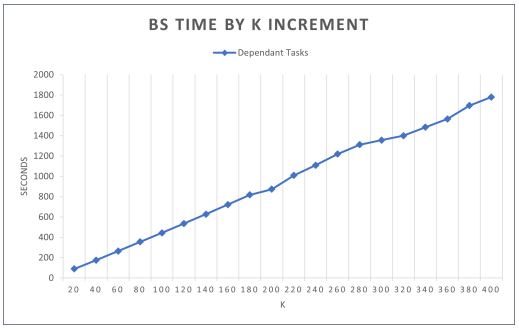
Beam Search (BS) is dependent on all the same factors that DFS but has the increasing difficulty of finding K best solutions at each step of the process, making the algorithm stronger but also slower. Given that time escalates much deeper in BS, some graphs have been taken with different number of points of observation and separated individually for a better representation.

First the impact of the chosen beam width (K) is observed. It is interesting to see how it

affects the time and the number of nodes it expands, as the algorithm will reach a point in which increasing the K will not produce better results as there are less possible nodes per Value than K.



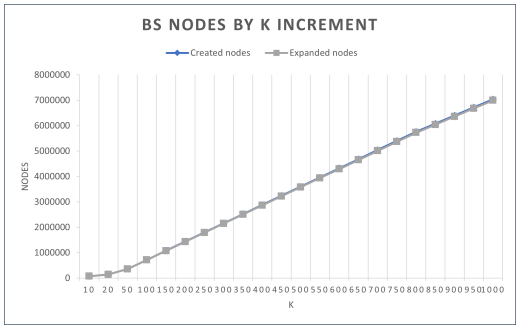
(a) Regular tasks



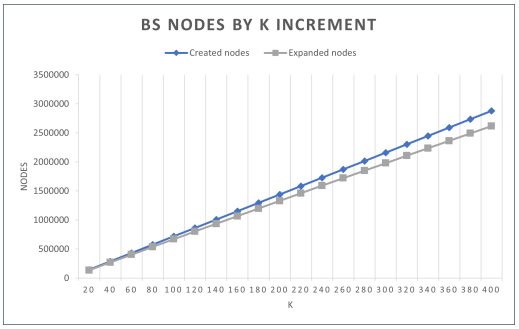
(b) Dependant tasks

Fig. 5.9. BS time by K increment

By increasing the K a polynomial growth of degree 2 in the time can be seen with regular Tasks whilst the growth is linear when Tasks are dependant. As the scale of the graph is higher with the dependant tasks, a K of 400 implies an algorithm time upwards of 30 minutes, whilst with regular tasks a K of 800, double, is required to reach such times.



(a) Regular tasks



(b) Dependant tasks

Fig. 5.10. BS nodes by K increment

As can be observed, the nodes also increase linearly with the beam width and in the case of non-dependant Tasks almost the same number of observed nodes are expanded. This is because it is harder to find compatible solutions if there is dependency, specially when it is hard coded as in the worst case in which the testing was realised. Still, expanding most of the created nodes means most of the solutions are being observed and thus a better solution is to be found.

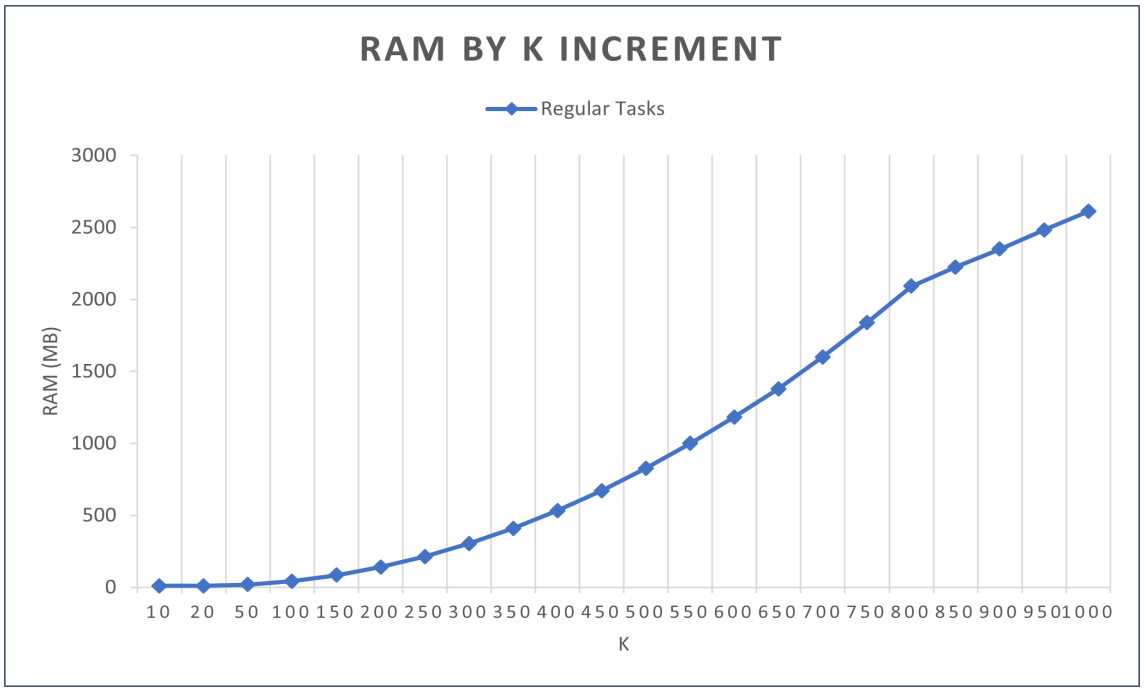
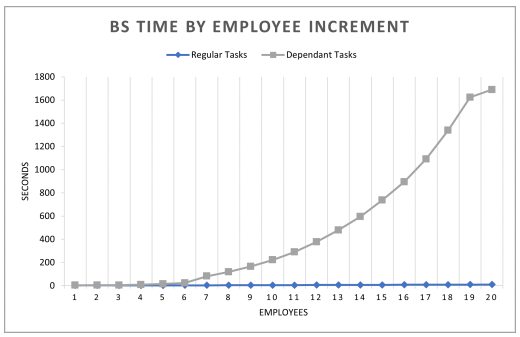


Fig. 5.11. RAM usage by K increment

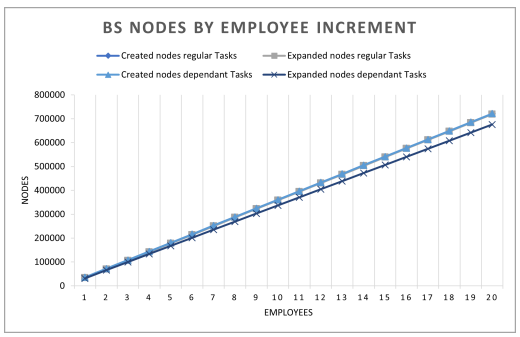
Lastly the increment in RAM usage is studied. It is important to note that even if this graph only shows the results with regular Tasks, the impact of dependant Tasks was just barely noticeable. The increase is thus polynomial of degree 4 reaching up to almost 3 gigabytes of RAM when K is set to 1000.

Given that good or perfect solutions are already found in the realm of beam widths of 100, the proper use of this value would be to execute the software increasing the K slightly a couple of times to observe if better solutions are found, as the overall execution time will not be particularly high in any case.

Now the graphs for the time and nodes as affected by the inputs will be described, similarly to the ones shown for the DFS algorithm.



(a) Regular tasks

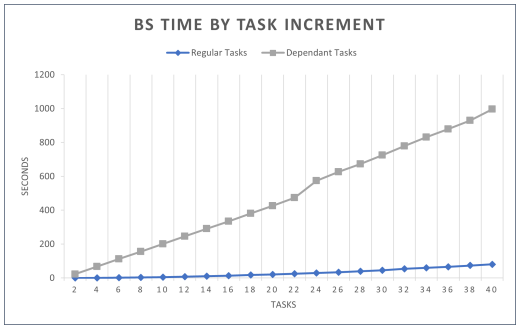


(b) Dependant tasks

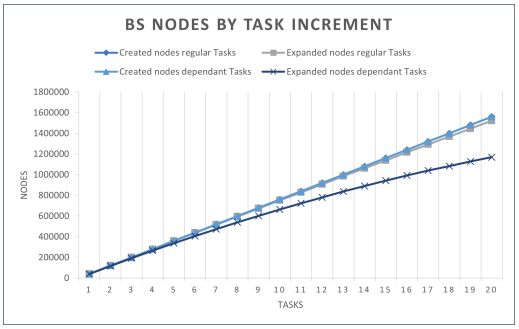
Fig. 5.12. BS by Employee increment

Results show that whilst time is barely unaffected by the increase in Employee count when the Tasks are regular, when they are dependant the time increase is highly noticeable, reaching almost 30 minutes with a count of just 20 Employees. This delay in time is due to the activation of the third type of mutex, that was nonexistent in the regular case and delays time further and further as there are more nodes to explore with the increase of the Values.

On the other hand, the nodes graph shows that under regular Tasks the number of created and expanded nodes is practically the same, that much so that both lines in the graph are superimposed. Introducing dependant Tasks does lower the number of expanded nodes rightfully so, as the possibilities decrease with the mutex inclusion.



(a) Regular tasks

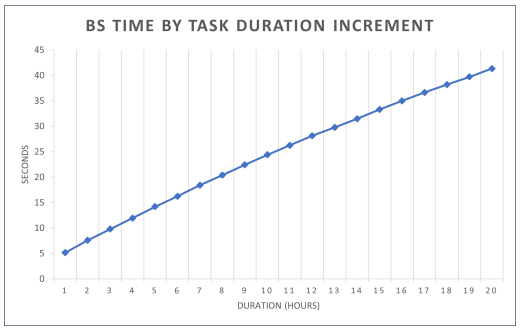


(b) Dependant tasks

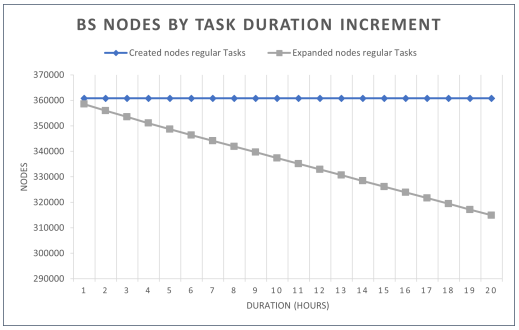
Fig. 5.13. BS by Task increment

These graphs show how increasing the Tasks has a much wider margin than the Employees. Regular Tasks barely increase the time whilst dependant ones have a linear growth slower than the one with Employees, meaning dependency of the Tasks is more reliant in the number of Employees that perform those Tasks, than in the number of Tasks themselves.

With the number of created and expanded nodes a similar curve to the one with Employees (Figure 5.12) can be seen but with fewer expanded nodes in the case of dependant Tasks. This again is to be expected as the number of nodes (width of the level) increases with Employees, not with Tasks, which increase the depth of the tree.



(a) Regular tasks

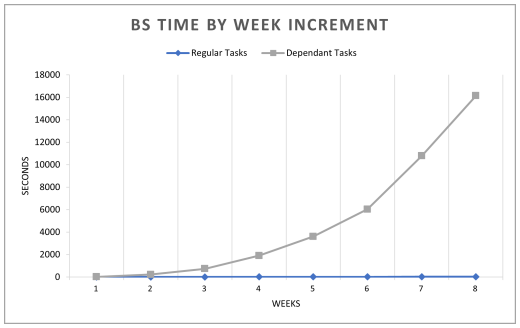


(b) Dependant tasks

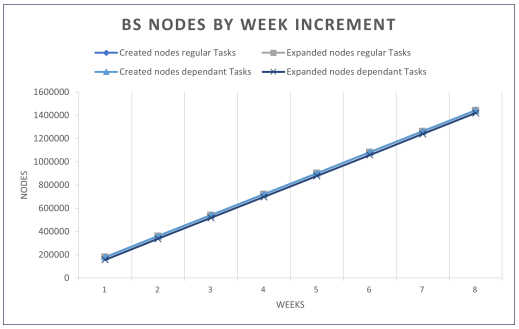
Fig. 5.14. BS by Task duration increment

Once again, Task duration does have an impact on the time just as was seen in Figure 5.4 with mutex and due to the same reason. Nodes on the other hand have a steady perfectly linear decline in number as the Task duration is increased. This behaviour is caused by a decrease in number of nodes as the duration of the task increases, as there is less and less margin for Tasks to start.

In other words, the last Tasks to possibly be started are created earlier and earlier in time halving the number of nodes by each unit the time is increased. If the last Task can start at Friday 16:00 with duration 1, it must start at Friday 15:00 with duration 2, meaning that whilst in the first case there were two possible nodes (Friday at 15:00 and 16:00), there is now only one (Friday at 15:00), reducing the number of nodes.



(a) Regular tasks



(b) Dependant tasks

Fig. 5.15. BS by week increment

Increasing weeks has only a marginal effect on time with regular Tasks but yields a massive increase when there is dependency. Being dependant tasks already one of the most defining factors as has already been seen in this section, once it is mixed with weeks yields almost exponential increases. Times can reach almost 5 hours when weeks are set to 8 which is the highest bottleneck in the whole software, and thus implies that weeks should be set to the minimum when executing it.

Looking at nodes, the graph shows that the created ones are the same regardless of the type

of Task but including dependency decreases the number slightly, still by a similar margin as in the previous graphs even if it cannot be seen perfectly due to the high numbers that are represented.

Once again it is important to take a look at a real world example and not only the worst case scenarios that are being discussed.

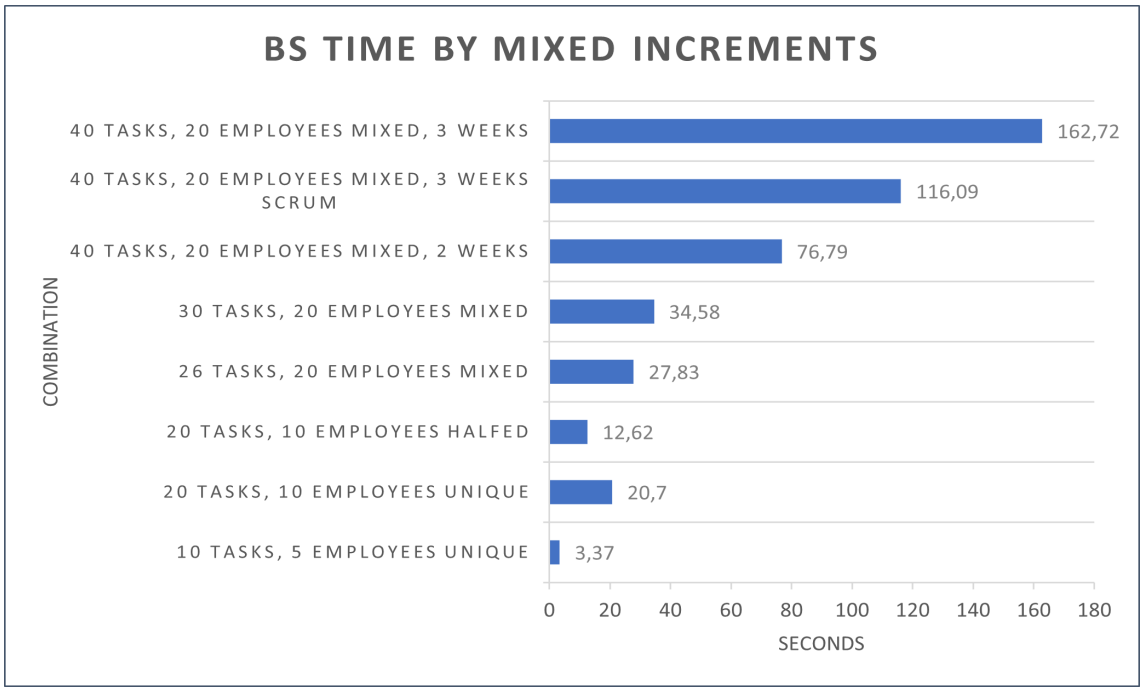


Fig. 5.16. BS time by mixed increments

As the graph suggests, once Task dependency is not enforced the Beam Search times are very mild even with a high number of input variables, as was seen in Figure 5.6. It is only when increasing the number of Tasks together with the weeks that the time grows significantly, still managing to complete the algorithm in less than 9 minutes which, compared to the results in the previous graphs, is a very good time frame.

Once all essential tests have been done, the heuristics that indicate how the K nodes are chosen (as detailed in section 4.3.3.2) are to be studied now. Whilst they all expand the same number of nodes and tend to achieve the same results, they do not necessarily take the same time, meaning there should be a prefered algorithm, the one that takes the less time to find the best solution.

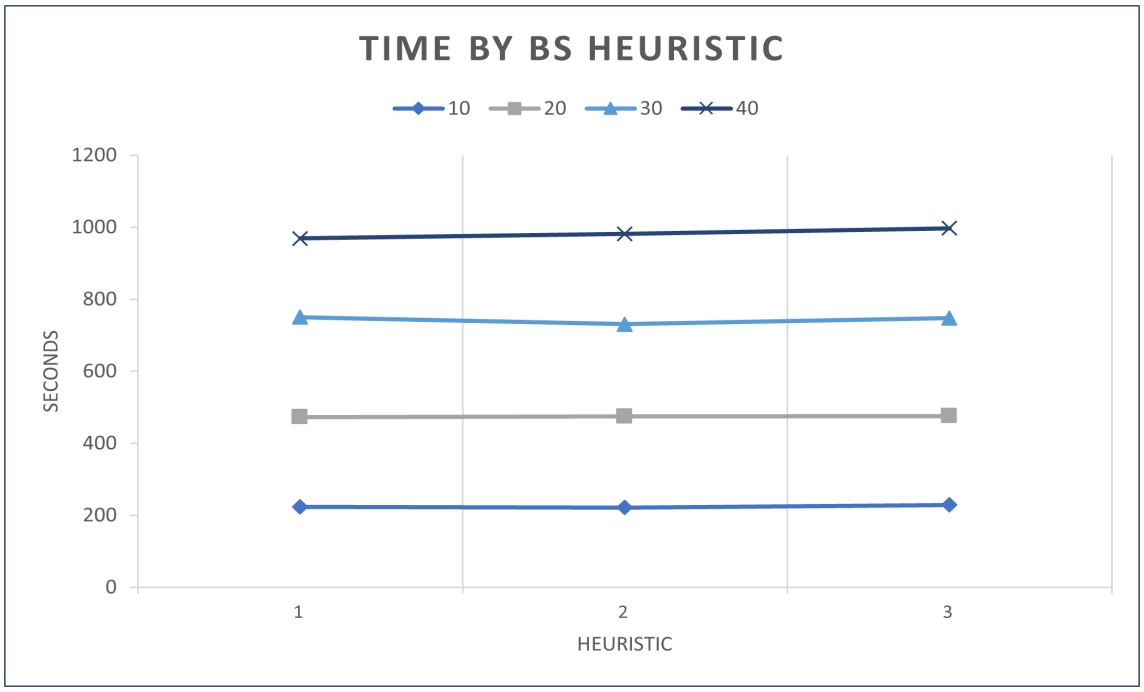


Fig. 5.17. BS time by beam width heuristic

Results show that time variation is within margin of error and thus the election of the algorithm is down to the quality of the solution it yields. This is surprising as heuristic 3 has the added cost of calculating the finishing time of each Task and thus should always be the most time consuming option, but as the results show, making that calculation does not affect time complexity.

Given that the software supports a high beam width while still maintaining good time frames, the election of the beam heuristic becomes irrelevant, which could be considered either a success in the design of the heuristics or a failure in finding an exceptionally good one.

5.2.3. Parallelism

Now that all results of the main parts of the software have been shown, the impact of the thread implementation in the mutex calculation and the parallelism of the sorting performed during the Beam Search will be observed.

The tests performed in this section were done with a constant setting of 10 Employees and an increasing number of non-dependant Tasks that is shown in the graph. The rest of the parameters are the ones indicated at the beginning of this chapter.

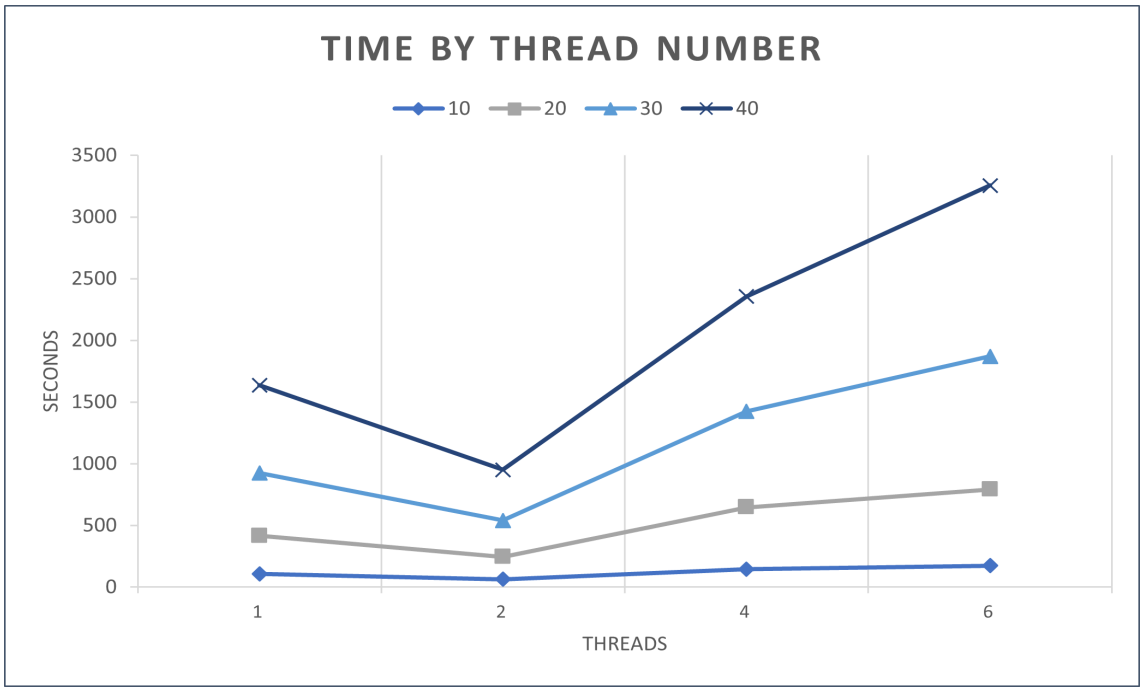


Fig. 5.18. Mutex time by thread increment

As can be observed, creating more threads can be counterproductive and the optimal point is achieved with only two of them. As the time difference is extremely noticeable, at points even reaching worse results than those achieved when increasing the Tasks, the possibility of the worsened results being caused by the overhead of creating threads is discarded.

Instead, the most probable explanation for the decrease in time quality is that increasing the number of threads is also increasing the readings to the sets that store the Values, meaning there is an overhead being generated that slows the whole execution. Still, being able to gain up to a 70% increase in speed by using just a single more core is a fine gain.

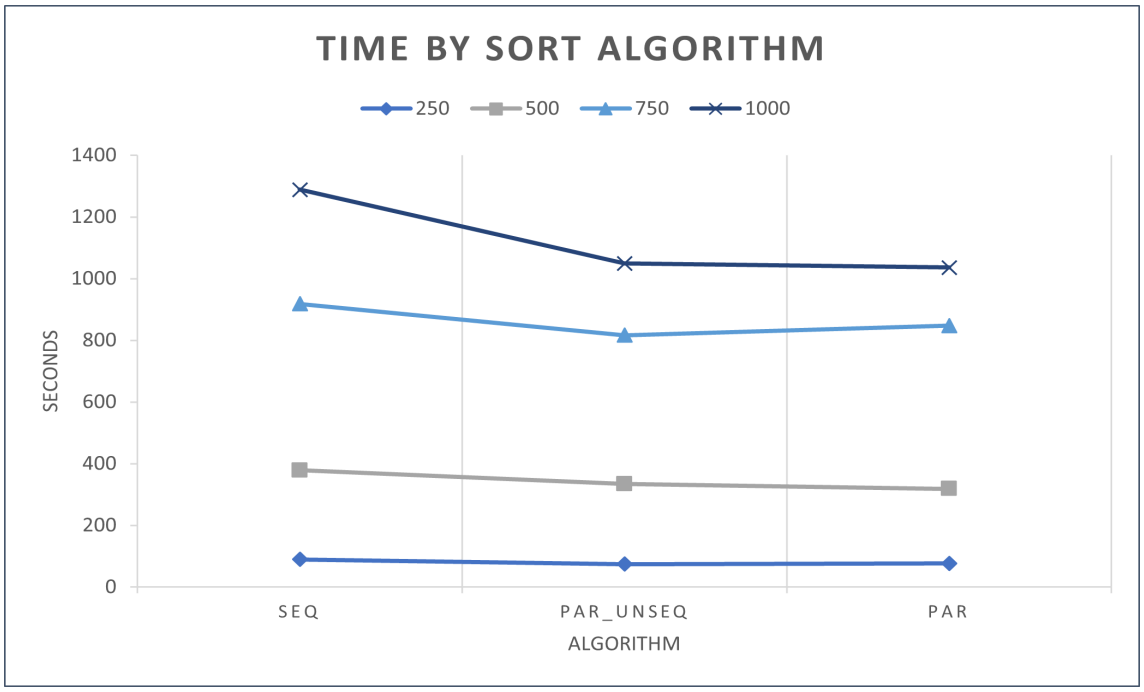
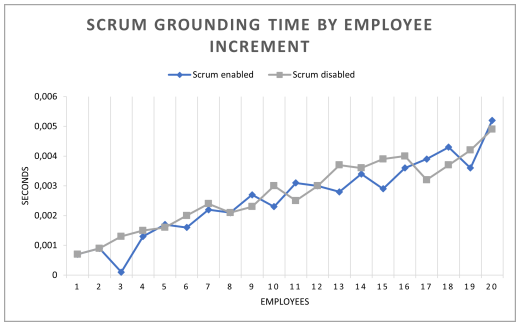


Fig. 5.19. BS time by sorting algorithm

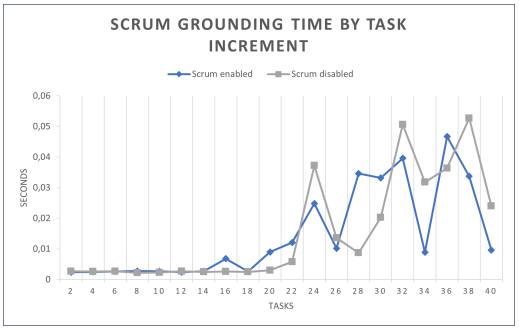
Comparing the different sorting algorithms does not produce results as differentiated as in the previous graph, but there is still a clear pattern. The more Tasks introduced the bigger the difference is between the sequential algorithm and the parallel ones, being both parallel algorithms practically the same in speed under all tests. This leads to the conclusion that setting parallelism is much preferred but that choosing one type over the over will lead to no noticeable gains, as suggested in the official Visual Studio guidelines [25].

5.3. Scrum results

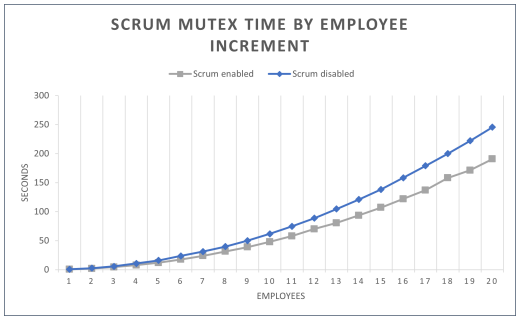
Lastly, activating the option for Scrum daily reunions reduces the number of total inputs by not creating all Tasks that start at that reunion time in the grounding process. This effectively reduces the total number of Values and thus the time the rest of the calculations take and the number of nodes the algorithms create and expand. The following graphs show this effects in detail:



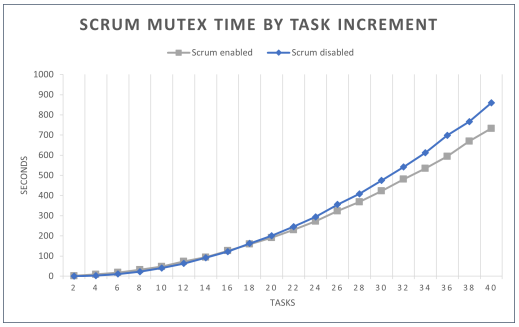
(a) Grounding time by Employee increment



(b) Grounding time by Task increment



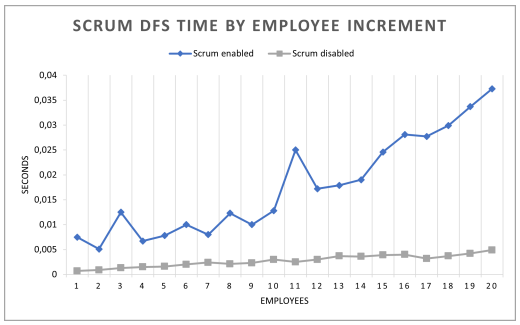
(c) Mutex time by Employee increment



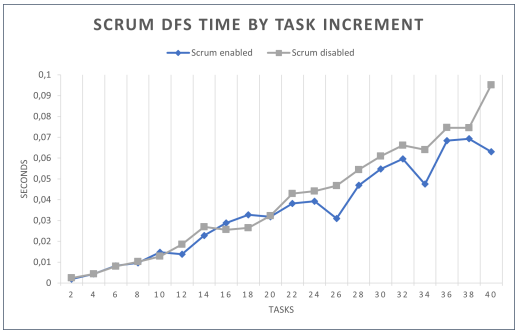
(d) Mutex time by Task increment increment

Fig. 5.20. Scrum Grounding and Mutex time

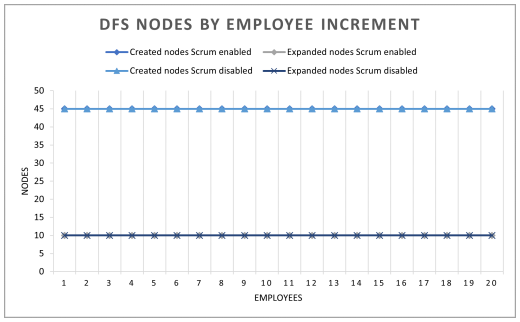
As expected, there is a polynomial decrease in time of up to 25% when increasing Tasks and up to 23% when increasing Employees that is not translated to the grounding time as the time frames are really small.



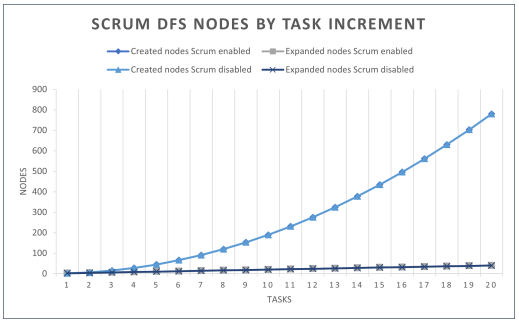
(a) DFS time by Employee increment



(b) DFS time by Task increment



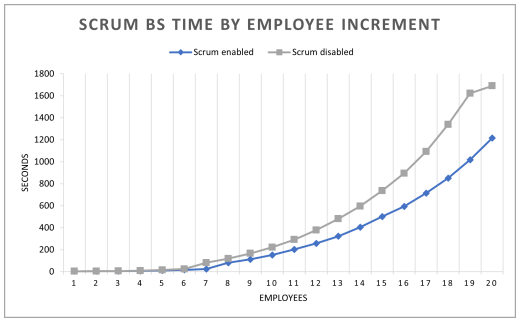
(c) DFS nodes by Employee increment



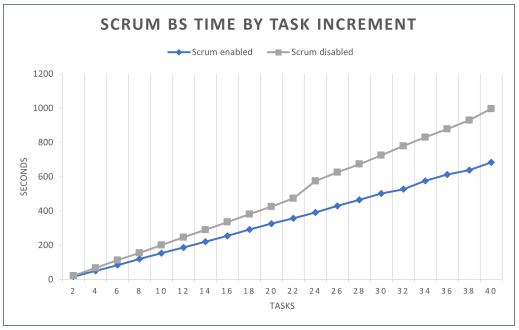
(d) DFS nodes Task increment increment

Fig. 5.21. Scrum DFS time and nodes

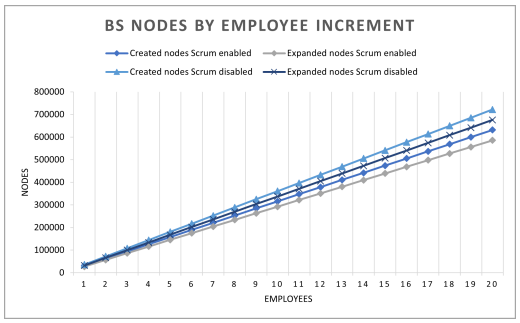
Looking at the Depth-First Search graphs shows that whilst this time there is a noticeable increase in time when there is an Employee increment, the contrary seems to happen when Tasks are increased. Looking at the nodes it is interesting that there was not a single change in the number of created nodes, meaning that the ones that were removed with the Scrum time were not a part of the first plausible solution and thus did not affect the execution of the algorithm.



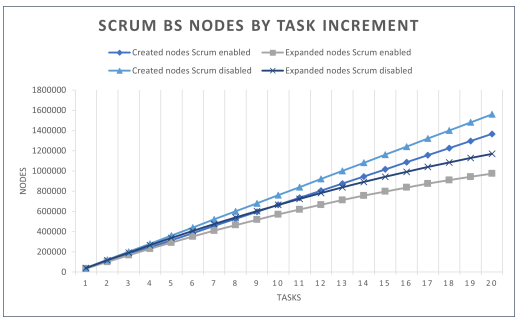
(a) BS time by Employee increment



(b) BS time by Task increment



(c) BS nodes by Employee increment



(d) BS nodes Task increment increment

Fig. 5.22. Scrum BS time and nodes

Similarly to what was seen in Figure 5.20 with mutex time, there is a polynomial decrease of grade 3 in time of up to 28% and 31% in the case of the Employee increment and Tasks increment respectively. Contrary to DFS, but as mentioned previously, the number of both created and expanded nodes has decreased as there are less possible Values that can be explored by the algorithm.

6. CONCLUSIONS

Now that all details of the project, its objectives, its implementation, and its testing have been discussed, the conclusions can be drawn. This section will be divided into three sections as there are several different areas to be discussed.

6.1. General conclusions

Based on the results, the speed of the software, and the quality of the solutions, it can be said that the project has been a success. The objectives were reached and while the Scrum implementation could be enhanced, my understanding of the C++ language and parallelism algorithms has been greatly increased.

It is true that this is not software meant for professional use as the lack of interface suggest, but any user with an understanding of basic programming skills can compile its own version for personal use.

From the results section it can be noted that there are very noticeable bottlenecks if any of the variables limit is exploited, but for the initial intended use that is Scrum Sprints, there is a healthy margin for experimentation with a high number of input variables.

Whilst time will greatly increase if the schedule covers several weeks, it is still a good time frame, specially considering the result is valid for months. The addition of parallelism and multi-threading has no doubt become an essential part of the software speed and offers still more headroom for improvement.

This is a specially good aspect of the whole project as there is still more work that can be done to enhance the characteristics and provide a good user experience that can expand the user base in the future.

6.2. Implementation conclusions

The use of a constraint satisfaction problem as a model and the two selected search algorithms as solvers has proven to be a great modeling structure to solve the problem I had stated for this project.

Given the results, the ideal use of the software would be to apply the Beam Search algorithm with a beam width of at least 100 using 2 threads and parallel sorting algorithms, which leads to great results no matter the applied beam width heuristic.

In this regard, it could be stated that all the preference heuristics are good or rather that the problem scope in a real world application is narrow enough that there is no need for more complex formulas. After all, the best solution will always be found at the leftmost

part of the search tree, instead of at the end of each branch, making the execution fast by nature.

This is because the earliest time is always desired, and can only be applied if the Values of each branch are in time order, as a random order would make the time non deterministic, invalidating most of the testing done in chapter 5. Thus, a logical order in the input files eases the solving of the problem. It is evident that the greatest bottlenecks are the week number and the dependency of the Tasks, meaning those numbers should be kept as low as possible, but regardless, good solutions were always found.

Regarding Depth-First Search, given the high number of Values and its scaling with the number of inputs, a complete implementation of the algorithm becomes inefficient as the solving time is extremely large. Thus, the implemented algorithm that only finds the first valid solution is only useful for an early examination of the problem that can give a very raw look at the solution and help estimate the weeks parameter for the then longer execution using Beam Search algorithm.

6.3. Encountered challenges

Throughout the development of the project there has been more than one instance where the progress slowed significantly and several weeks or days were spent solely on the same issue with no real advancements in the overall project.

Starting by the formulation of the design and the class diagram, there was a month were no design was valid and thus the Depth-First Search implementation was not feasible. In the end, it was a matter of translating the literal interpretation of a Constraint Satisfaction Problem to code, modeling the variables accordingly, but the trial and error proved helpful if only to get more familiarized with the environment and the C++ language.

While testing the search algorithms solutions kept having slight imperfections such as miss-calculations of the Task duration or incorrect dependency appliance. While some of these could have been solved with a more detailed design of the mutex types into code, it also helped polish significantly the Task duration calculation algorithm and ensure all the found solutions are always correct even if not necessarily optimal.

7. PROJECT PLANNING AND BUDGET

This chapter will cover the costs of developing this software, the expected impact in the society and economy, and its possible use cases.

7.1. Project planning

The project spanned a total of 245 days since it was registered in the university data, up until the writing of these document. The detail planification is shown in the following graph:

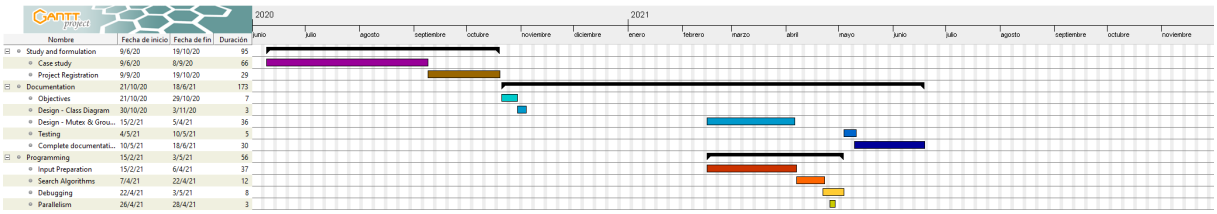


Fig. 7.1. Gantt diagram

- **Study and formulation**

  For a couple of months the project theme changed a lot from a study of software optimizations in the form of threads and parallelism to what is the current version of it, focused on a scheduling problem. Once the subject was clear, and the professor approved it, it was registered.

- **Objectives**

  The main structure of the project was formalized and studied with the professor as a form of 'contract' for what the software would be and what it should contain. This took only a couple of weeks as the points were really clear.

- **Design**

  The class structure, software functions, and utilities were designed in advance to establish a baseline from which start the programming. The main algorithms design and investigation was done at this point and took several months as programming for it was done at the same time and finding the correct modeling for the input data was troublesome.

- **Input preparation programming**

  All the programming for the software sections that involved preparation for the search algorithms, including input parsing and modeling, Grounding, and mutex.

– **Search algorithms programming**

  The main programming for the two search algorithms, the treatment of the found solutions, and the logging of them in the output form. This took only a couple of weeks as I could focus completely on it without any other responsibilities.

– **Parallelism**

  After the algorithms were completed, parallelism was implemented ensuring the results were not altered and only the speed was improved. As C++17 already included parallelism algorithms for sorting, this step only took a few days to implement and polish threads for mutex.

– **Debugging**

  Whilst the main programming was done, a lot of debugging was made to ensure the found solutions were optimal and correct.

– **Testing**

  Before proceeding to the final documentation steps, the tests were performed and all the gathered information was formatted accordingly.

– **Documentation**

  Finally, the writing of this document was done and the project determined finalized. This took the final weeks of the project as there was a lot of time available and the quality was to be perfect.

7.2. Project budget and costs

With the planning in mind, the project budget and its cost of completion can be calculated. The cost will be broken down into the 35 weeks that the project took to make, first calculating the material costs and then the human resources cost divided by sections.

As the whole software was developed using a desktop computer and a laptop for commuting, these are the total material costs including additional peripherals:

Material	Price (€)
Desktop computer	1450,00€
Laptop computer	1100,00€
Monitors	470,00€
Keyboard	110,00€
Mouse	70,00€
Total cost	3200,00€

Table 7.1. MATERIAL COSTS

Even though the project was develop by a single person, in a real case there would have been more people involved. As a modern computer engineer has an education in many more areas than just programming, many of the roles in a standard company can be achieved by a developer with no additional resources needed.

The required roles with their standard salary in Spain have been obtained from the Hays Guide for the Labour Market [28] using the 0-2 years of experience metric in Madrid. Assuming 52 weeks with 5 labour days, 14 days of festivities [29], and 23 vacation days [30] yields the following calculation (1784 hours of work per year):

- Project leader: A project leader with a salary of 50.000€ per year earns 28,02€ per hour.
- Programmer: A c++ programmer in Madrid has a salary of 28.000€ per year, earning 15,69€ per hour.
- Tester: A standard tester in a QA department has a salary of 30.000€ per year, earning 16,81€ per hour.

Thus, for a development time of approximately 137 hours not counting the time spent in the "Study and formulation case", the total cost is as follows:

Role	Salary (€)
Project Leader	3.838,74€
C++ Programmer	2.149,53€
QA Tester	2.302,97€
Total cost	8.291,24€

Table 7.2. HUMAN RESOURCES COSTS

Combining the material costs and the human resources costs, the total expenditure of the project is reached:

Resource	Salary (€)
Material resources	3200,00€
Human resources	8.291,24€
Total cost	11.491,24€

Table 7.3. TOTAL PROJECT COSTS

8. REGULATION AND SOCIO-ECONOMIC IMPACT

This chapter will cover how this project may affect the current economic environment, whether that is by simply releasing the software or by its manipulation. It will also detail the current regulation and technological standards that may be applicable to the use of the software.

8.1. Socio-Economic Impact

The release of this project as a open-source software allows any business, whether big or small, to make use of the tool to estimate their workflows. Whilst there are other alternatives that may be easier to use and contain an intuitive interface, they may not supply the same features this software does and may require an expensive license to be used in a professional environment.

Ideally the scheduler would be used to help estimate a project duration before it has begun. If the developed project is in-house, the scope and tasks of it should be found beforehand, and if it depends on a client a meeting is necessary to find its necessities. Essentially, the requirements need to be established before the execution of the software, as it is necessary to estimate the duration of the tasks before using it.

Once the software has been used, the results can be presented to the client to provide an estimated duration of the whole project and a detailed planning of its development. As said previously, its intended to be used several times during the development, as inevitably tasks may take longer or be accomplished in a shorter time. This also makes a good use of the Scrum Sprints, as a new estimation can be done before each client meeting with the new results according to the Sprint performance.

As the software output is a CSV file, additional tools such as the Gantt Project [31] can be used to provide a visual representation of the schedule similar to the one used in section 7.1. It can also simply be used together with other similar project planners to provide additional needed features or information.

The main goal is for the software to serve as an estimation tool that can help alleviate the workflow of the project by providing a good schedule ahead of time. Having the whole project planned before its beginning avoids situations where new unplanned tasks surface or development takes longer than what was promised leading to delays.

8.2. Regulation

Patent infringement and legislation concerns need to be considered when releasing software to the public, even it it is open source like this project.

This software is free to use and distribute under open-source conditions, and thus it is not intended to be patented.

The use of the software does not guarantee the optimality of the result and the correct use of the provided schedule. An incorrect interpretation or employment of the result may be performed by the user of the software, it is thus recommended to review the results before putting them to use.

An input of Employees and Tasks is required for the software to work properly, but it is not logged or processed in any way, meaning the L.O. 3/2018 of December 5 [32] and Regulation (EU) 2016/679 [33] are not applicable. The data that is processed is only parsed into usable forms and then transformed into the solutions, it is not modified.

The software detailed in this document was developed from scratch and without the use of any possibly copyrighted code. Other free software was used in the development of the project and this document that does not incur a fee or a license. The software is the following:

- Libre Office 7.1.3 [34]
- C++ 17 programming language [16]
- Microsoft Visual Studio Community 2019 Version 16.9.3 [35]
- Overleaf online LaTeXeditor [17]
- Gantt project [31]

9. FUTURE WORK

The completion of this software has been focused on achieving all the objectives detailed in chapter 3 and given the time constraints, some things were left out of the table. This section will cover future improvements that can be done to the software to increase its performance and expand its features.

9.1. Increased customization of dates

The current implementation of the software only allows for weeks with five labour days as the calculation of the finishing time of the Tasks is heavily reliant on manual algorithms. With the introduction of C++20 and its new Calendar system [36], these algorithms can be replaced with new functions that make them more flexible, fast, and automatically adaptative to different number of labour days.

As businesses introduce four labour day weeks, these companies have seen an improvement in production and employee happiness as an additional rest day per week has proven to be extremely beneficial. Being able to schedule projects with customizable labour days would allow these kind of settings to be compared to regular ones and then pitched to the people in charge to suggest their implementation.

9.2. GPU accelerated parallelism

Whilst investigating ways of parallelizing the algorithms of C++, I stumbled across GPU accelerated parallelism. Nvidia offers its own completely fleshed out library and compiler for C++ that allows parallel algorithms included in C++17 to be executed by the CUDA cores of their graphics cards [37].

This compiler is fairly easy to use but requires a Linux environment, thus its out of the scope of the current project that I decided to limit to the Visual Studio environment of Windows. This is because I run into problems compiling the current version of the code in Linux, concretely with the use of the C++17 parallel algorithms as they required special compiler libraries in Linux.

9.3. Further Beam Search parallelization

Whilst the mutex implementation makes use of threads, the Beam Search algorithm is entirely dependant on the C++17 included algorithms for the parallelism. Introducing threads in Beam Search would allow for further parallelization of the search and increased speeds.

This could be done in two different ways:

- Splitting the whole tree search into N threads, essentially dividing the K nodes into those threads and calling the algorithm that number of times. When those threads are finished the K best results from all the threads would be selected. This would be similar to the mutex thread use.
- Splitting the K nodes of each branch (Variable) into N threads and selecting the K best results of all the threads before proceeding to the next branch. Essentially creating and deleting N threads in each branch.

It would be interesting to see whether the creation and deletion of many threads in the second approach affects the performance heavily, and whether the results of the additional threads in the first approach worsen the results. It would also be necessary to take into account the already use of threads in the sorting with the C++17 algorithms, as those threads may conflict with the ones in use.

9.4. Interface

Currently the software can only be used by manually editing the input settings and directories in the code. This inherently limits the scope of the project and its usefulness for users that are not familiarized with programming or professional environments that need a more robust interface.

Implementing an interface should become a priority in the case that the software was to be commercialized or advertised for a broader audience. It would also be necessary if it was adapted for the special requirements of a potential client that may want a product more adjusted for its use case.

9.5. Additional implementations

As said previously, this software is fairly focused on the scheduling of projects for teams, and cannot be used for other kinds of businesses. Including time limitations on the Tasks would make the project valid for use cases such as the scheduling of clients that can only be attended at specific times.

The addition of this feature would mean altering the design of the main variables as well as the implementation of the grounding and mutex, but it would also alleviate a lot the dependence on calculating the duration of the Tasks, as these kind of works are time limited.

BIBLIOGRAPHY

[1] D. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*, Illustrated. Blue Hole Press, 2010.

[2] K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*, 1st ed. Financial Times Prentice Hall, 2002.

[3] J. Sutherland, D. Patel, C. Casanave, J. Miller, and G. Hollowell, *Business Object Design and Implementation*, 1st ed. Springer-Verlag London, 1995.

[4] K. Schwaber and J. Sutherland. (2020). "The scrum guide," [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[5] D. Rechter, *Constraint Processing*. Morgan Kaufmann, 2003.

[6] E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, 2014.

[7] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley Longman Publishing Co, 1984.

[8] S. Edelkamp and S. Schrodl, *Heuristic Search*, 1st. Morgan Kaufmann, 2011.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[10] *A* diagram*. [Online]. Available: https://stackoverflow.com/questions/5849667/a-search-algorithm.

[11] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, pp. 497–520, 1960.

[12] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewel, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[13] *Branch & bound diagram*. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/2/20/Branch%26bound.jpg.

[14] F.S.Cooper *et al.*, "Speech understanding systems: Report of a steering committee," *Artificial Intelligence*, vol. 9, pp. 307–316, 1977.

[15] *Beam search diagram*. [Online]. Available: https://www.semanticscholar.org/paper/A-new-Hybrid-Filtered-Beam-Search-algorithm-for-of-Mej%C3%ADa-Ni%C3%B1o/2a27cdfa0e344aecbacb4f13439fb46f2c581584/figure/3.

[16] ISO/IEC. (2017). "Working draft, standard for programming language c++17," [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf.

[17] *Overleaf online LaTeXeditor*. [Online]. Available: https://www.overleaf.com.

[18] T. Hahmann. (2011). "Constraint satisfaction problems (backtracking search)," [Online]. Available: http://www.cs.toronto.edu/~torsten/csc384-f11/lectures/csc384f11-Lecture04-BacktrackingSearch.pdf.

[19] V. K. Chaudhri. (2011). "Constraint satisfaction problems," [Online]. Available: https://web.stanford.edu/class/cs227/Lectures/lec14.pdf.

[20] H. Ghaderi. (2011). "Backtracking search (csps)," [Online]. Available: http://www.cs.toronto.edu/~hojjat/384w09/Lectures/Lecture-04-Backtracking-Search.pdf.

[21] J. T. Point. (2018). "Informed search algorithms," [Online]. Available: https://www.javatpoint.com/ai-informed-search-algorithms.

[22] Reducible. (2020). "Depth first search (dfs) explained: Algorithm, examples, and code," [Online]. Available: https://www.youtube.com/watch?v=PMMc4VsIacU.

[23] J. Brownlee. (2018). "How to implement a beam search decoder for natural language processing," [Online]. Available: https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/.

[24] A. Williams, *C++ Concurrency in Action*, 1st ed. Shelter Island, New York: Manning Publications Co, 2012.

[25] B. O'Neal. (2018). "Using c++17 parallel algorithms for better performance," [Online]. Available: https://devblogs.microsoft.com/cppblog/using-c17-parallel-algorithms-for-better-performance/.

[26] R. Schulung. (2018). "C++ core guidelines: Rules for concurrency and parallelism," [Online]. Available: https://www.modernescpp.com/index.php/c-core-guidelines-rules-for-concurrency-and-parallelism.

[27] B. Filipek. (2017). "C++17 in details: Parallel algorithms," [Online]. Available: https://www.bfilipek.com/2017/08/cpp17-details-parallel.html.

[28] Hays. (2021). "Guía del mercado laboral 2021," [Online]. Available: https://www.hays.es/documents/63345/4314146/Hays+-+Gu%c3%ada+del+Mercado+Laboral+2021.pdf.

[29] B. O. de la Comunidad de Madrid. (2020). "Decreto 80/2020, de 23 de septiembre, del consejo de gobierno, por el que se establecen las fiestas laborales para el año 2021 en la comunidad de madrid.," [Online]. Available: https://www.comunidad.madrid/sites/default/files/doc/empleo/bocm-calendario_laboral_comunidad_de_madrid_2021.pdf.

[30] B. O. del Estado. (2015). “Real decreto legislativo 2/2015, de 23 de octubre, por el que se aprueba el texto refundido de la ley del estatuto de los trabajadores,” [Online]. Available: https://www.boe.es/boe/dias/2015/10/24/pdfs/BOE-A-2015-11430.pdf.

[31] *Gantt project*. [Online]. Available: https://www.ganttproject.biz/.

[32] ——, (2018). “Ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales.,” [Online]. Available: https://www.boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf.

[33] O. J. of the European Union. (2016). “Regulation (eu) 2016/679 of the european parliament and of the council,” [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN.

[34] *Libre office*. [Online]. Available: https://libreoffice.org/.

[35] Microsoft. (2019). “Visual studio community 2019,” [Online]. Available: https://visualstudio.microsoft.com/es/.

[36] ISO/IEC. (2020). “Working draft, standard for programming language c++20,” [Online]. Available: http://open-std.org/jtc1/sc22/wg21/docs/papers/2020/n4861.pdf.

[37] D. Olsen, G. Lopez, and B. A. Lelbach. (2020). “Accelerating standard c++ with gpus using stdpar,” [Online]. Available: https://developer.nvidia.com/blog/accelerating-standard-c-with-gpus-using-stdpar/.

APPENDIX A. SOLUTION SAMPLE

Introduction

To better understand this document an example of an application of the software is provided in this section. The number of inputs, their characteristics, and the settings of the algorithms have been selected appropriately for the best coherence and variance. The intent of this example is to serve as an indication of a real world application of the developed software.

Input details

The selected parameters are the following:

- Tasks: 40.
- Employees: 20.
- Schedule: 9:00-17:00 Monday to Friday.
- Weeks: 3.
- Threads: 2.
- K: 50.
- BS algorithm: Algorithm 2.
- Sort algorithm: par_unseq.
- Daily Scrum time: 11

Name	Role
Rodrigo	CEO
Arturo	Programmer
Sergio	Programmer
Luis	Programmer
Daniel	Programmer
Víctor	Programmer
Julia	Lawyer
Alejandro	Lawyer
Roberto	Designer
Andrea	Designer
Paula	Designer
Guillermo	Accounting
Isabel	Accounting
Cristina	Accounting
Javier	HR
Alicia	HR
Juan	HR
Lucía	Marketing
Álvaro	Marketing
Jacinto	Marketing

Table 9.1. EMPLOYEES INPUT CSV FILE

Name	Duration	Compatibility	Dependencies
Concept	8	CEO	-
Environmental Design	10	Programmer	Concept
Character Design	9	Designer	Concept
Level Design	15	Designer	Concept
Music Alignment	18	Designer	Concept
Music Ambient	10	Designer	Concept
Music Late Motifs	14	Designer	Concept
Combat Programming	15	Programmer	Music Late Motifs|Level Design
World Programming	10	Programmer	Concept
Inventory Programming	10	Programmer	Concept
Music Testing	7	Programmer	Music Late Motifs
Programming Testing	10	Programmer	Inventory Programming
Debugging	19	Programmer	World Programming|Combat Programming |Inventory Programming
Design Testing	12	Programmer	Level Design
Legal Preparation	15	Lawyer	Concept
Copyright Research	7	Lawyer	Concept
Patent Infringement	8	Lawyer	Concept
Contract Revision	11	Lawyer	Concept
IP Registration	12	Lawyer	Concept
Competition	6	Marketing	Concept
Advertisements	15	Marketing	Promotional Material
Influencer Contracts	20	Marketing	Concept
Promotional Materials	14	Marketing	Concept
Media Liaison	15	Marketing	Concept
Recruitment Campaign	18	Marketing	Concept
Social Networks	15	Marketing	Concept
Salary payments	8	HR	SS Contracts
Documentation	14	HR	Concept
IRPF Management	10	HR	SS Contracts
SS Contracts	10	HR	Concept
Recruitment	12	HR	Concept
Health and Safety	7	HR	Concept
Expansion Investigation	8	HR	Concept
Monthly Expenditure	7	Accounting	Payrolls
Investments Payoff	21	Accounting	Concept
Future Viability	10	Accounting	Concept
Cost Reducing	14	Accounting	Monthly Expenditure
Real Estate Investigation	22	Accounting	Concept
Tax Returns	17	Accounting	Concept
Payrolls	14	Accounting	Concept

Table 9.2. TASKS INPUT CSV FILE

Execution time

Compiling the code led to these results for the parameters:

Section	Time (seconds)	
Grounding	0,014347	
Mutex	501,58	
DFS	0,196568	
BS	162,719	
Total time	664,51	
Algorithm	Created nodes	Expanded nodes
Grounding	1747	40
Mutex	768120	610917

Table 9.3. RESULTS WITHOUT SCRUM

Section	Time (seconds)	
Grounding	0,0158	
Mutex	395,30	
DFS	0,1379	
BS	116,09	
Total time	511,54	
Algorithm	Created nodes	Expanded nodes
Grounding	1741	40
Mutex	672105	515946

Table 9.4. RESULTS WITH SCRUM

Solution

Employee	Task	Date	Duration
Rodrigo	Concept	22/02/21 09:00	8
All employees	Daily Scrum	22/02/21 11:00	1
Arturo	Environmental Design	23/02/21 10:00	10
Roberto	Character Design	23/02/21 10:00	9
Paula	Level Design	23/02/21 10:00	15
Andrea	Music Alignment	23/02/21 10:00	18
Víctor	World Programming	23/02/21 10:00	10
Luis	Inventory Programming	23/02/21 10:00	10
Julia	Legal Preparation	23/02/21 10:00	15
Alejandro	Copyright Research	23/02/21 10:00	7
Lucía	Competition	23/02/21 10:00	6
Álvaro	Influencer Contracts	23/02/21 10:00	20
Jacinto	Promotional Materials	23/02/21 10:00	14
Javier	Documentation	23/02/21 10:00	14
Juan	SS Contracts	23/02/21 10:00	10
Alicia	Recruitment	23/02/21 10:00	12
Guillermo	Investments Payoff	23/02/21 10:00	21
Isabel	Future Viability	23/02/21 10:00	10
Cristina	Real Estate Investigation	23/02/21 10:00	22
All employees	Daily Scrum	23/02/21 11:00	1
Lucía	Media Liaison	24/02/21 09:00	15
Alejandro	Patent Infringement	24/02/21 10:00	8
All employees	Daily Scrum	24/02/21 11:00	1
Roberto	Music Ambient	24/02/21 13:00	10
Arturo	Programming Testing	24/02/21 14:00	10
Juan	Health and Safety	24/02/21 14:00	7
Isabel	Tax Returns	24/02/21 14:00	17
Alicia	Expansion Investigation	24/02/21 16:00	8
Jacinto	Recruitment Campaign	25/02/21 10:00	18
Javier	Salary payments	25/02/21 10:00	8
All employees	Daily Scrum	25/02/21 11:00	1
Paula	Music Late Motifs	25/02/21 12:00	14
Sergio	Design Testing	25/02/21 12:00	12
Julia	Contract Revision	25/02/21 12:00	11
Alejandro	IP Registration	25/02/21 12:00	12
Juan	IRPF Management	25/02/21 14:00	10
Álvaro	Social Networks	25/02/21 16:00	15

Employee	Task	Date	Duration
Guillermo	Payrolls	26/02/21 10:00	14
Lucía	Advertisements	26/02/21 10:00	15
All employees	Daily Scrum	26/02/21 11:00	1
All employees	Daily Scrum	01/03/21 11:00	1
Arturo	Combat Programming	01/03/21 12:00	15
Sergio	Music Testing	01/03/21 12:00	7
Guillermo	Monthly Expenditure	02/03/21 10:00	7
All employees	Daily Scrum	02/03/21 11:00	1
Guillermo	Cost Reducing	03/03/21 10:00	14
All employees	Daily Scrum	03/03/21 11:00	1
Arturo	Debugging	03/03/21 13:00	19
All employees	Daily Scrum	04/03/21 11:00	1
All employees	Daily Scrum	05/03/21 11:00	1
-	Finish	08/03/21 10:00	-

Table 9.5. SCHEDULER SOLUTION