

# 5LIG0 – Applied Combinatorial Algorithms 2024/2025

## Assignment 1: Volvo CE automatic quarry site

Due date: November 25 (part 1) / December 9 (all parts)

---

### 1 Purpose

In this assignment, you will create solutions for three vehicle routing problems inspired by an industrial use case. This covers the following course-level ILOs:

1. explain the concepts of divide-and-conquer, greediness, dynamic programming, backtracking, branch-and-bound and approximation, and
2. model the complexity of algorithms by using the big- $\mathcal{O}$  method, and
3. develop an algorithm to solve a combinatorial-optimization problem using appropriate concepts (see ILO 1).

**Skills:** The purpose of this assignment is to help you practice the following skills that are essential to your success in this course / in school / in this field / in professional life beyond school:

- Applying existing state-of-the-art concepts and algorithms to a graph-based optimization problem. Graphs are a common abstraction in this field, and being able to work with this abstraction (in terms of formalization, algorithms, and complexity analysis) therefore is highly relevant.
- Reporting on your work and properly presenting results and conclusions. This clearly is useful to practice because you will need this often in your professional life.

**Knowledge:** This assignment will also help you to become familiar with the following important content knowledge in this discipline:

- Shortest-path algorithms on graphs, concepts for optimization (i.e., greediness, branch-and-bound), worst-case complexity analysis.
- Implementations in a programming language of choice to solve the optimization problem.

## 2 Problem description

### 2.1 Introduction

Volvo Construction Equipment (Volvo CE) is a major international company, a subsidiary of the Volvo Group, that is specialized in developing, manufacturing, and marketing equipment for construction and related industries. This assignment focuses on one of the projects of Volvo CE at an electric quarry site in which different types of gravel are produced.

Fig. 1 shows a picture of the quarry site. The material (gravel) is produced by a primary crusher machine at the *loading points*. Then it is moved by autonomous truck *haulers* to a secondary crusher machine *unloading points*, avoiding *static obstacles* and other haulers. The haulers are loaded by a primary crusher machine or a human-operated wheel loader. The autonomous haulers are electrical and can be recharged at multiple *charging stations* on the site.

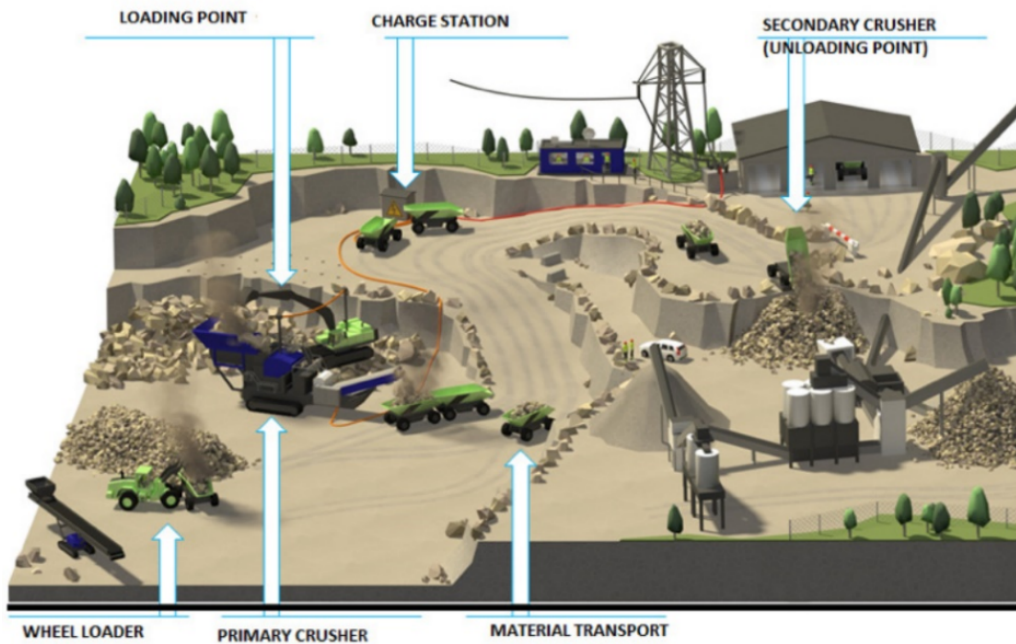


Figure 1: Volvo CE automatic quarry<sup>1</sup>.

For simplicity, we consider the site area as a table of  $12 \times 12$  blocks so that each region of the site is covered by a block with a specific coordinate given in the format  $[x, y]$ , where  $x$  is the horizontal block number (ranging from 1 to 12), and  $y$  is the vertical block number (ranging from 1 to 12).

#### Attention

Although the dimension of the area is  $12 \times 12$  in the examples below, you must assume that the width and height may vary to accommodate future sites (e.g., sites which are modeled by a  $120 \times 80$  grid). This is relevant, for instance, for your complexity analysis.

An example configuration is shown in Fig. 2. In this example, a cell with a black background is a static obstacle (**SO**) (obviously, haulers cannot pass through these obstacles). The loading and unloading points are indicated by **LP** and **ULP** respectively. The single charging station





<sup>1</sup><https://www.volvoce.com/global/en/news-and-events/press-releases/2018/testing-begins-at-worlds-first-emission-free-quarry/>

	1	2	3	4	5	6	7	8	9	10	11	12
1										ULP1		
2												
3												
4		LP1										
5												
6												
7												
8												
9												
10												
11												
12												

Figure 2: An example quarry site.

is indicated by the **power-plug** symbol. A hauler has a **mission** that requires it to visit several LPs and ULPs in a certain order. The hauler finishes its mission when it reaches its last destination. Haulers can move vertically or horizontally but not diagonally. It is assumed that **movement to a neighboring cell takes 1 second**.

A hauler can occupy empty cells, and cells with a loading point, unloading point, or charging station. It cannot occupy a cell with a static obstacle, or a cell with another hauler (that results in a **collision**). For simplicity, we consider that the time is discrete and that two haulers can cross each other without collision. For example, they can switch positions in two neighboring cells without collision:

	1	2	3	In this case, the haulers can switch their positions without any collision.
1				
	1	2	3	If both haulers want to enter the cell [2, 1] in their next step, a collision happens since both haulers will be in the same position after the movement.
1				

A **problem instance** has one or more haulers that each have a mission to be completed under a varying set of constraints. The **makespan** of the problem instance is the maximum of the mission durations of the individual haulers. The assignment consists of three parts that are described next.

### Part 1: Single hauler, unlimited battery

This part is the simplest version of the problem. We assume that there is only a single hauler and that the hauler does not need to be recharged during its mission (it has an unlimited battery).

### Part 2: Single hauler, limited battery

In this part of the assignment, we consider a single hauler with a limited battery capacity. We assume that the hauler has been equipped with a battery with (i) a *maximum capacity* and (ii) a certain amount of *initial stored energy*. If the battery runs out, then the hauler cannot move anymore and the mission has failed. Moving to a neighboring cell (horizontally or vertically) takes 50 units of energy. Since the hauler's mission can take a considerable distance to be completed, it may require visiting a charging station (maybe more than once per mission). We assume that the charging always takes 5 seconds.

### Part 3: Multiple haulers, unlimited battery

As you may have noticed, putting a lot of money on a site with only one hauler machine is not cost-efficient. To improve efficiency, we need to increase the degree of parallelism, i.e., the number of hauler machines. In this part, we have multiple haulers on the site to work in parallel, each with its own mission. Since the haulers work in the same area, they should avoid running into each other (avoid collisions). Remember that this means that at any point in time, no two haulers can be in the same cell. As in part 1, we assume that all haulers have an unlimited battery and thus there is no need for charging.

The general goal of each part of the assignment is to find a path for each hauler that realizes the mission of that hauler and that respects the given constraints: *your solution must be functionally correct*.

Two important quality metrics that you need to take into account are (i) the makespan (i.e., do you have an *optimal* solution or not), and (ii) the time needed to compute a solution.

## 2.2 Encoding of problem instances

There is a general input format for all parts of the assignment. A problem instance is defined by two files: `config.txt` and `mission.txt`. The following template explains the input format for the configuration of a problem instance (in the `config.txt` file):

```

1 Int      // Number of haulers (for example 1)
2 Int      // Number of LPs (for example 2)
3 Int      // Number of ULPs (for example 2)
4 Int      // Number of SOs (for example 10)
5 Int      // Number of charging stations (for example 0)
6 1*2 Int vector // Initial position of the haulers (for example [12,5])
7 List of 1*2 Int vectors // Position of the LPs (for example [8,9]-[3,10])
8 List of 1*2 Int vectors // Position of the ULPs (for example [5,2]-[6,11])
9 List of 1*2 Int vectors // Position of the SOs (for example [4,4]-[1,9])
10 List of 1*2 Int vectors // Pos. of the charging station (for example [7,3])
11 Int      // Maximum capacity of the hauler battery (for example 5000)
12 Int      // Initial stored energy in the hauler battery (for example 3500)

```

It is important to note that if some parameters are not needed for a specific part, you can ignore them after reading the input file. For example, in part 1 there is no charging station in the environment.

A hauler's mission is defined as a sequence of transportations between loading points (LPs) and unloading points (ULPs) and it is specified by a vector for each hauler like the one below (these are in the `mission.txt` file):

```
1 L2,U1,L3,U3,L1,U3      // mission vector for the hauler 1
2 U1,L1,U2,L3,U1,L2      // mission vector for the hauler 2
```

We have prepared a dataset of problem instances for you. We call this dataset **Dataset1** and you can find it on Canvas along with the other provided materials for the assignment. For each part of the assignment, there are 90 problem instances in Dataset1 that are structured as follows:

- 30 easy scenarios in which there are only a few static obstacles,
- 30 medium scenarios in which there are more static obstacles, and
- 30 hard scenarios in which there are many static obstacles and walls.

Each of these 30 scenarios is again divided into 5 categories based on the mission length from 4 to 14 with a step of 2. This is because longer missions are typically more difficult than shorter missions.

### Hint

It may help you, especially with the more difficult parts of the assignment, to define your own grid and problem instances to test your solution on corner cases. E.g., using a  $3 \times 3$  grid with two haulers to explore corner cases for part 3. Note that comments at the end of a line are separated by a tab (`'\t'`) and `'//'` characters.

## 2.3 Encoding of solutions

For each input problem instance, you need to produce an output file that shows the result of your solution. This file should be a `.txt` file and have the following format:

```
1 // Quantitative values
2 Int // Makespan (max of all mission completion times)
3 Int // Mission completion time hauler 1
4 Int // Mission completion time hauler 2
5 Int // Application execution time (in milliseconds)
6 // Path to the final points formatted as:
7 // time,hauler1position,hauler2position,...,haulerNposition
8 Int,[x,y] pair,[x,y] pair,...,[x,y] pair
```

An example with two haulers:

```
1 // Quantitative values
2 81 // Makespan (max of all mission completion times)
3 72 // Mission completion time hauler 1
4 81 // Mission completion time hauler 2
5 490 // Application execution time (in milliseconds)
6 // Position of each hauler for every (discrete) time step
7 0,[11,9],[8,12]
8 1,[11,8],[8,11]
```

```

9 2, [11, 7], [8, 10]
10 .
11 .
12 .
13 81, [2, 3], [6, 4]

```

### Hint

- ☺ Since the input and output formats are very important for the final evaluation, we have attached a few examples in the attachment of the assignment description on Canvas.
- ☺ The overall generated path indicates the position of each hauler at every time slot from time 1 to the time that the last mission is completed. Therefore, if you decide to stop a hauler for collision avoidance reasons, the next cell must be the same as the current cell. Similarly, a hauler must stay for 5 seconds at a charging station to fully charge its battery.
- ☺ The running time of your solution is the time *after reading the input file* until the time you start to write the output file. Please store your final result (moves of the haulers) in a temporary data structure and when the last hauler finishes the mission, then create and fill the output file. This is to avoid the overhead of writing to a file during your running your algorithm.

### Attention

When a hauler has finished its mission, it disappears from the map (it cannot cause any collision with other haulers). However, to keep the consistency of the output file, its last recorded position should be repeated in the output file until the mission is complete for all haulers. For example, if there are two haulers and the second hauler finishes its mission at time 3 while the makespan is 5, the last position of the second hauler should be repeated until the end of the output file:

```

1 // Quantitative values
2 5 // Makespan (max of all mission completion times)
3 5 // Mission completion time hauler 1
4 3 // Mission completion time hauler 2
5 1 // Application execution time (in milliseconds)
6 // Position of each hauler for every (discrete) time step
7 0, [11, 9], [8, 12]
8 1, [11, 8], [8, 11]
9 2, [11, 7], [8, 10]
10 3, [10, 7], [8, 9]
11 4, [9, 7], [8, 9]
12 5, [8, 7], [8, 9]
13

```

### 3 Tasks for completing the assignment

To complete the assignment, you need to do the following:

1. For each part of the assignment, develop an algorithm using algorithmic concepts that have been introduced during the course.
2. Implement your algorithms in a language of choice (we suggest C/C++ or Python) and test them by means of Dataset1 for *functional correctness* and *makespan and performance*.
3. Report on your work. This includes a discussion of your solution (algorithmic concepts used), a worst-case complexity analysis, a discussion on the solution quality (i.e., optimal or not), and an evaluation using Dataset1.

This assignment has two deadlines:

- The first deadline is for completing *at least* part 1 of the assignment (both code and report). You will get feedback on this that helps you to improve your work
- The second deadline is for completing the full assignment. This is the work that will be used for grading.

We provide the following materials:

- Dataset1 consists of 90 problem instances for each part of the assignment.
- Examples of input and output files, including the baseline makespan and computation times for Dataset1.
- Python and C++ code skeletons, and a number of scripts to test, e.g., the functional correctness of your solutions.

### 4 Criteria for success

1. You deliver a PDF report in double-column IEEE format (see, e.g., <https://www.ieee.org/conferences/publishing/templates.html>; these are also in Overleaf):

- ☐ **PR:** It must be clear and concise, and must not exceed 8 pages.
- ☐ **MOD:** It must contain a section in which you explain how you have modeled your problem as a graph and discuss and justify the used data structures.
- ☐ **SOL:** For each part, your report must contain a sub-section in which you explain your solution approach including the algorithmic concepts used, and justify your algorithmic choices. You must discuss the functional correctness and relate this to the results (sanity checks) on Dataset1. You can use, e.g., diagrams or pseudocode to clarify your approach.
- ☐ **OPT:** For each part, your report must contain a sub-section in which you discuss whether or not your algorithm is optimal. Include a formal proof of optimality (in case you believe it is optimal; reuse well-known results from the literature), or a counter example that shows that the algorithm is not optimal.



- **CPX:** For each part, your report must contain a sub-section in which you explain and justify the worst-case complexity of your solution using Big- $\mathcal{O}$  notation. Note: the parameters of the complexity equation must be parameters of the input problem, for example,  $\mathcal{O}(E + V)$  is wrong because  $E$  and  $V$  are not parameters of the input problem. On the other hand, the width and height of the area (which may vary), the number of items in the mission vector, and the number of haulers are examples of parameters of the input problem.
- **RES:** For each part, your report must contain a sub-section in which you discuss the performance (both makespan and computation time) with respect to a baseline implementation that we provide. Finally, there is a discussion on the relation with the complexity analysis that you did. This sub-section is based on the problem instances of Dataset1.

## 2. You deliver code and results

- Store your code which includes your solutions in a folder and add a subfolder per part.
- Your code should execute using usual C/C++ compilers (e.g. `gcc/g++` if you use C/C++), or you should make the compilation instruction clear to the reviewer (if you are using a different compiler or programming language). You can find a simple template folder for the assignment on Canvas.
- Store the output file (per problem instance in Dataset1) for each part of the assignment separately (using the same folder structure as the input files in Dataset1).
- Provide one `.csv` file that includes a summary of your output results next to the baseline result (in terms of makespan) using the scripts provided for you.

### Hint

- ☺ Model the problem as a graph and apply algorithms that you have seen in the lectures.
- ☺ You can reuse your solution for part 1 in the next parts.
- ☺ For part 2 you can choose an approach that gives an optimal solution, or an approach that does not. Justify your choice. You can also do both and discuss the trade-off (this is excellent, but not expected).
- ☺ An optimal solution for part 3 is hard to find, and we do not expect you to discover it. Your solution needs to be functionally correct, however!
- ☺ Report on the makespan and running time of your algorithms on Dataset1 by using a boxplot, see Fig. 3 for an example. On the vertical axis of the diagram, you can plot the *normalized makespan* or the running time of your solution. Normalized makespan is the ratio between the makespan of your solution to the makespan of the baseline solution.
- ☺ Note that in order to get timely feedback, you should contact the TA and meet them during the reserved slots on Wednesdays (indicated by TA hours) in the course schedule. Do not wait until the week before submission (because it might become too late to correct your whole solution in the last week).



- ☺ We suggest that you use C/C++/Python to implement the code, but you are free to code in any other programming language as well. If you are not using C/C++/Python, please give a heads-up to the TAs so that they can prepare the environment to compile and execute your code when they want to test your algorithm. *A code template is provided for those students who decide to develop their solution in C++ or Python.*
- ☺ The standard [gcc](#) compiler should be able to compile your C++ code. For Windows users, we recommend [Code::Blocks](#) IDE.

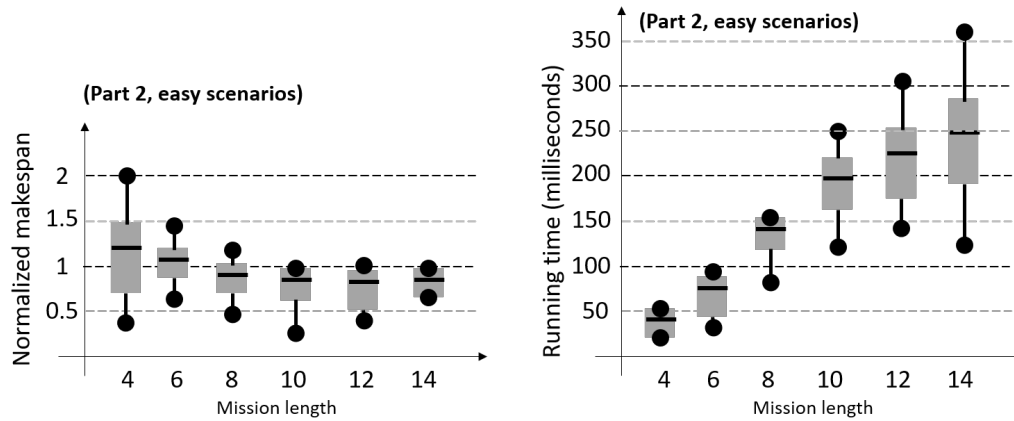


Figure 3: An example of charts for reporting the performance.