

Applied Combinatorial Algorithms (5LIG0)

ASSIGNMENT 1: VOLVO CE AUTOMATIC QUARRY SITE

Weight: 40%

(v2.0)

2022

1 Assignment Overview

System-of-systems, or collaborating systems, consist of several interdependent systems that collaborate to reach a common goal. Intelligent transportation systems and automated guided vehicles are typical examples of collaborative systems. In this assignment, we focus on traffic management systems that provide routing services for individual vehicles in an autonomous transportation system with the goal of effective traffic management. Other examples of traffic management systems can be found in air traffic control, railroad scheduling, and the traffic management of unmanned aerial vehicles and autonomous transport vehicles.

This assignment considers a vehicle routing problem from an industrial use case (Volvo quarry site) to improve fuel and energy consumption, environmental costs, and system efficiency. This allows you to practice with and apply your knowledge about graph-based algorithms to find optimal collision-free routes for autonomous vehicles.

1.1 Learning objectives

At the end of this assignment, you will be able to model a vehicle routing problem as a graph problem, apply algorithmic techniques from the course to make new algorithms for optimal routing, collision-free routing, and path planning under restricted energy/battery constraints. You will also be able to explain your solution, argue about its computational complexity, and convince your audience that your solution is effective (results in good solutions) and efficient (has a reasonable complexity or runtime).

1.2 Deadline

The final deadline for the assignment is Dec. 13th at 23:59 (see Canvas).

1.3 Assessment

The assignment will be assessed based on the quality of the solution, depth of the discussions and explanations provided in the report as well as in the oral exam, and the quality of the report. To know more about criteria that impact the grade of this assignment, see the questions in Assignment 2 (peer review). Those questions provide you a guideline on how your report will be assessed.

1.4 Structure

The assignment has several parts. It is suggested that you finish the easier parts within the first week (after studying the graph analysis lectures) and then spend the rest of the time on the last two parts.

1.5 Communication, feedback, and correctness of the solutions

You are encouraged to discuss your solution approach with the TAs and receive feedback **before submission** to ensure that your solution is correct (incorrect solutions will result in failing the course). Moreover, we have provided some scripts to check the sanity of your final output files. These scripts will identify if your solution violates the constraints (for example, a vehicle crashes into walls or another vehicle, or a vehicle does not complete its mission). If a part of your solution fails these ‘sanity checks’, then you will not get the points of that part of your solution.

Note that in order to get a timely feedback, you should contact the TAs 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).

1.6 Rules and important notes

This is an individual assignment. Collaboration and discussion with fellow students are allowed, but you are supposed to submit your own, individual report, output results, and your individually written code. All reports and code will be checked for plagiarism (i.e., presenting the work of others as your own). Duplicate sections of the report or code will be detected, even if modified. If you use code segments or templates from other sources, then credit those sources. If you develop solutions together with fellow students, insert a short paragraph in your report describing the collaboration and your individual contributions.

1.7 Hints and guidelines

Read the problem description very carefully and try to understand the problem first before designing your solution. Try to make your solution efficient. For each part of this assignment, you will need to design an algorithm, analyze the complexity of it, implement it, test it, evaluate it on the input data provided for you and produce the output file according to the given format, write a report that explains these steps, and your conclusions.

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.

Notice

The standard [GCC](#) compiler should be able to compile your C++ code. For Windows users, we recommend [Code::Blocks](#) IDE.

2 Provided materials

We provide the following materials:

- An example of input/output format
- A dataset (called **Dataset1**) of 90 test scenarios for each part of the assignment (except for Part 1 which is 30 scenarios)
- Two code templates for C++ and Python to read input files and write output files. If you develop your solution in another programming language, you need to implement these functions by yourself.
- The result of our implementation on the problem instances of **Dataset1** to be used as the **baseline**. Your solution should be as good as or better than the baseline algorithm.
- We will also provide some scripts to generate output files for all given input files, perform a sanity check on the output files, and produce a file that compares your solution with the baseline method. To run that script, your code must be able to provide one output file per test case (a config file and a mission file). The scripts and their documentation will be uploaded soon on Canvas.

3 Deliverables

1. Report

- It must be clear and concise.
- It should contain a sub-section in which you **explain how you have modeled your problem as a graph** (explain what is represented by vertices and edges).
- For each part, you must explain and **justify your choices and support them by the evaluation results**. When is your algorithm efficient and when is it not (and why)?
- Your report should contain a separate sub-section for each part describing whether or not your algorithm is optimal. **Include a formal proof of optimality** (in case you believe it is optimal).
- **Explain and justify the complexity of your solution using Big O notation.** Note: the parameters of the complexity equation must be parameters of the input problem, for example, $O(E + V)$ is wrong because E and V are not parameters of the input problem but the width and height of the area, number of items in the mission vector, and the number of haulers are parameters of the input problem.
- The report should be delivered in IEEE format. The template is attached to the assignment. The report should not be more than 10 pages (unless you have figures, etc.).

2. Code and results

- Store your code which includes your solutions in a folder and add a **sub-folder 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.

4 Case Study: Volvo Construction Equipment Electrical Site

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. 4-1 shows a picture of the quarry site. The material (gravel) is produced by a primary crusher machine (*PCR*) at the loading points (*LP*). Then it is moved by autonomous truck haulers (known as *HX* machines introduced by Volvo CE) to a secondary crusher machine. The *HX* machines are loaded by *PCR* or a human operated wheel loader (*WL*). The autonomous *HX* machines are electrical and can be recharged at multiple charging stations (*CH*) on the site.

In the following, we start with a simple version of the scenario in Part 1 and then make the problem more challenging step by step in the next parts.

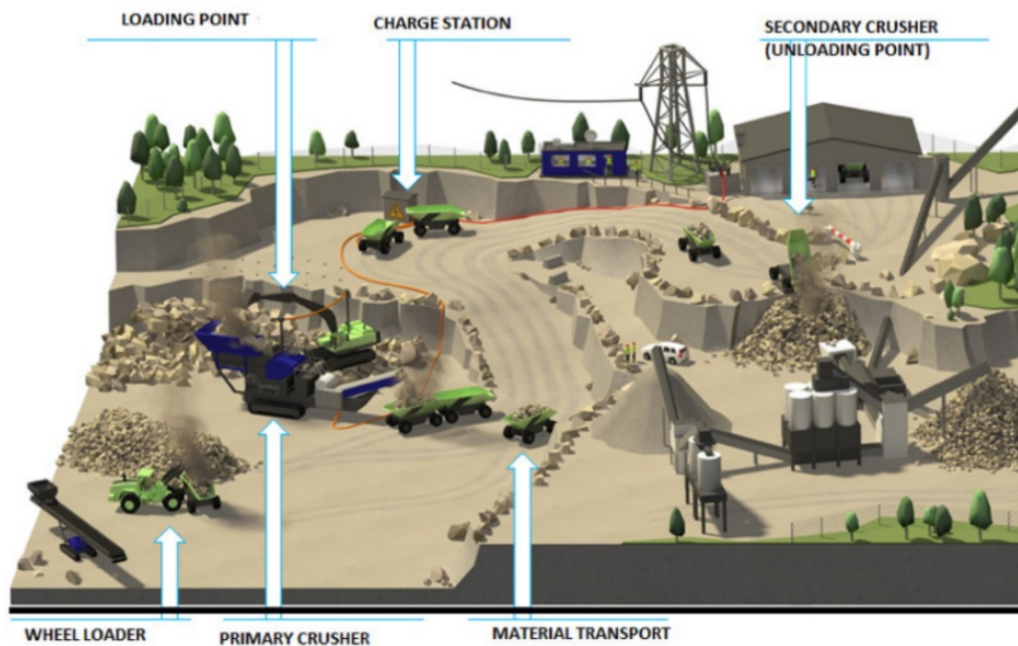


Figure 4-1: Volvo CE automatic quarry¹

4.1 General problem specifications





For simplicity, we consider the site area as a table of 12×12 blocks so that each region of the site is covered by a block with a specific coordinate – given in the format **[vertical_block_number, horizontal_block_number]**. Each block can be occupied by one object at most (except for loading and unloading points which can contain the hauler machine besides crushers or wheel loaders). An example configuration is shown below. In this example, cells with a black background are static obstacles (vehicles cannot pass through these obstacles). **LPs** and **ULPs** are the loading and unloading points. A hauler has a mission which requires it to visit a set of LPs and ULPs in a certain order. The hauler finishes its mission when it reaches to its last destination and then disappears from the map. Haulers can move vertically or horizontally but not diagonally.

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

Table 1: An example of quarry site

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

At any moment in time, only one object (hauler) can occupy a cell, namely, two haulers cannot be at the same position at the same time. 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				

4.2 Input format

There is a general input format for all parts of the assignment. This is to increase the usability of the source code especially the reading function. The following template explains the input format:

```

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 static obstacles (SOs) (for example [4,4]-[1,9]-[1,7])
10 List of 1*2 Int vectors //Position of the charging station (for example [7,3])
11 Int //Maximum capacity of the hauler's battery (for example 5000)
12 Int //Initial stored energy in the hauler's 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 Part1 there is no charging station in the environment or there is no static obstacle.

4.3 Hauler's mission format

The 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:

```

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

```

Notice

Comments after the parameters are separated by a tab ('`\t`') and '`//'` characters.

4.4 General output format

For each input problem instance, you will 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
3 Int // Mission completion time hauler 1
4 Int // Mission completion time hauler 2
5 Int // Application execution time (in millisecond)
6 // Path to the final points
7 time,hauler1position,hauler2position,...,haulerNposition
8 Int,1*2 vector,1*2 vector,...,1*2 vector

```

An example:

```

1 //Quantitative values
2 81 // Makespan
3 72 // Mission completion time hauler 1
4 81 // Mission completion time hauler 2
5 490 // Application execution time (in millisecond)
6 //Path to the final points
7 1,[11,9],[8,12],...,[7,11]
8 2,[11,8],[8,11],...,[7,10]
9 3,[11,7],[8,10],...,[7,9]
10 .
11 .
12 .
13 81,[2,3],[6,4],...,[3,5]

```

Notice

Like input files, comments after the parameters are separated by a tab ('`\t`') and '`//'` characters.

Notice

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.

Notice

The overall generated path indicates the position of each hauler at every time slot from time 1 to the time that the last mission completes. Therefore, if you decide to stop a hauler for collision avoidance reasons, the next cell must be the same as the current cell.

Notice

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 counting for the overhead of writing to a file in the running time of your solution.

4.5 Test sets (Dataset1)

To have a better insight into your solution and its performance/quality, we have prepared a dataset of test cases 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 (except for Part 1), there are 90 test cases in Dataset1 that are structured as follows:

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

These 30 scenarios are 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 complex than shorter missions.

We would like you to evaluate the performance of your solution for each class of scenarios (easy, medium, hard) using diagrams whose horizontal axis is the number of items in the mission (4, 6, 8, 10, 12, and 14). For each of these scenarios, there are 5 test cases in Dataset1. On the vertical axis of the diagram, you can plot various things, for example, the **normalized makespan** (the time at which the last mission completes) 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. We suggest that you use a boxplot, similar to the figure below. A box plot shows the minimum, maximum, median, and the third and first quartiles of the data (there are 5 test cases per mission length).

Please avoid only reporting the “average” values because then we will not see the **maximum** and **minimum** differences between the performance of your solution and the baseline method for those 5 test cases.

We also have a second dataset, called **Dataset2** which we do not share with you. We will use that dataset to assess the quality of your solution and to see if the results reported for **Dataset1** are somehow consistent with **Dataset2**. After you submit your solution, we will run your solution on the second dataset and share the results with the student who reviews your solution.

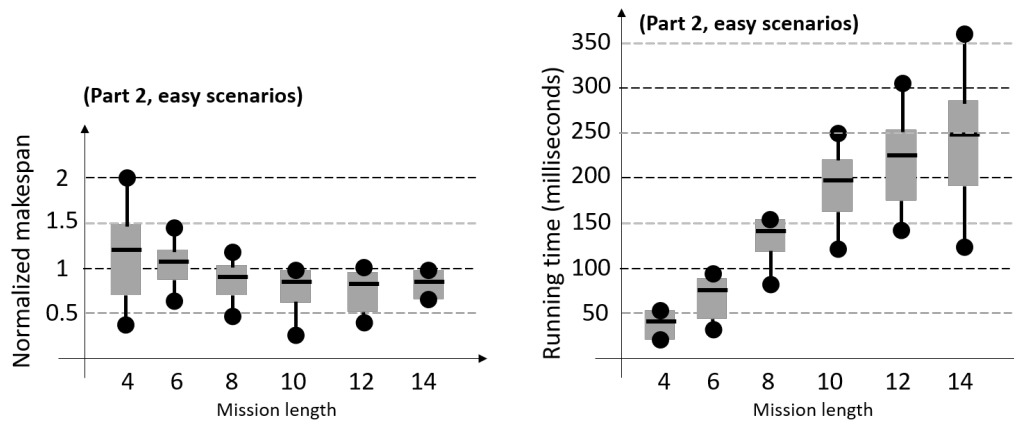



Figure 4-2: An example of charts for reporting the performance

4.6 Part 1: Single hauler, multiple loading and unloading points, no obstacles

This part is the simplest version of the problem. Assume that there is only one autonomous hauler on the site, responsible to carry material from different loading points (LP) to different unloading points (ULP). In this part, there are no static obstacles.

	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	ULP1						LP2					

Assumptions. We assume that the hauler machine does not need to be recharged during its mission (using an unlimited battery). Therefore, there is no need to define a charging station on the site.


Goals (requested output). Considering a movement time of 1 second for moving from one cell to any of its (vertical or horizontal) neighboring cells, the goal is to find the optimal path that leads to the minimum completion time of the assigned mission to the hauler.

Hint

Model the problem as a graph (explain this phase in your report) and find the shortest path for the hauler. We encourage you to keep your eyes open to other path-finding algorithms (such as A* algorithm) which might not have been covered in the course.

4.7 Part 2: Single hauler, multiple loading and unloading points, static obstacles

In this part, apart from the assumptions of the first part, we have several static obstacles (SO) are added to the area. The configuration file includes the position of the static obstacles as well. An updated version of the previous example considering the static obstacles has been shown below.

	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	ULP1						LP2					

Goal. Like the previous part, the goal is to find the shortest mission completion time.

Hint

You may think of static obstacles in a graph representation as missing vertices/edges among the accessible vertices.

Suggestion. The remaining parts of the assignment will use the outputs of this part. Try to store it as a function and re-use it in the next parts.

4.8 Part 3: Multiple haulers, static obstacles, collision free minimum mission time

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 add a new hauler to the site to work in parallel with the first one. Since the haulers work in the same area, they should avoid running into each other (avoid collision).

The assumptions for this part of the assignment are the same as Part 2 (there are static obstacles, etc.). In addition, the input file will contain information on the initial position of other haulers (in the same way as it contains the position of the first hauler). The haulers' missions are different, but they are defined using the same file format.



Goal. The goal is to complete all missions as soon as possible. The time at which the last hauler finishes is called the **makespan**. At any point in time, no two haulers should be in the same cell (avoid collisions).

Hint

There are multiple ways to solve this problem, one of them is to use a greedy algorithm to decide which hauler must stop if there is going to be a collision. Of course, you can think of an optimal solution that always results in a minimum makespan. Proposing such an optimal solution (with the proof of optimality) means having an **outstanding solution** for this part (and there will be a reward for it)!

4.9 Part 4: Single hauler, static obstacles, charging station

As it was mentioned in the problem description, the haulers are autonomous machines moved by electrical motors which are powered by a battery mounted on the hauler. In this part of the assignment, we will consider haulers with a limited battery capacity.

	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	ULP1						LP2					

Assumptions. Keeping all assumptions from Part 3, in this part, we assume that each hauler has been equipped with a battery with specific properties such as maximum capacity, initial stored energy and charging time per second. It is also assumed that the consumed battery is indicated for movement per meter (which in this assignment is 10 units of battery per meter).

The distance between every two neighboring cells on the site is 5 meters (vertical, or horizontal neighbors). Since the hauler's mission can take a considerable amount of time to be completed, it requires to visit a charging station (maybe more than once per mission). There is one charging station on the site that can be accessed by a hauler at a time. For simplicity, we assume that the charging time always takes 5 seconds.

Goal. Find the optimal path for the single hauler such that it has the minimum completion time (considering obstacle avoidance and battery consumption).

Hint

Before designing your solution, think about the following questions:

- How/when do you decide to interrupt the hauler's mission and redirect it to the charging station?
- How do you guarantee that the hauler gets to the charging station "before" it is totally discharged? (a fully discharged hauler will not move anymore!)
- How reasonable is it to redirect the hauler to the charging station even if it has enough battery to reach the next LP/ULP block?

This problem has an optimal solution. You can think about finding the required shortest paths between the cells (not necessarily all), and then use them in a **binary decision tree** to see if the hauler should or should not go to the charging station between every two items of the mission (e.g., between LP1 and ULP2). You will need to develop some extra functions to calculate the shortest distances, consumed energy, and remaining energy of the hauler.

To improve the complexity of your solution, you may consider using a branch-and-bound method; find a way to cut branches that will not lead to an optimal result, for example, if the hauler is full, there is no point in considering going to a charging station during the next X items of the mission (note, the energy consumptions between mission items is not equal).