Optimizing the Home Care Service

Project 2 IT3708 Spring 2024

Groups Allowed? Max 2 persons. Every student must attend the demo day. The group members should sign up for the same time slot on demo day.
Guidance? Questions will be answered during lab hours or on the Blackboard forum.
Deadline: 15th of March

Introduction

In this project, you will implement a genetic algorithm (GA) to solve a simplified version of a real-life optimization problem. It involves creating routes for home-care nurses in order to visit and care for patients while minimizing time spent driving. Solving this problem has a great social impact, as nurses have more time for patient care and spend less time planning.

Visma Resolve is involved as a guest project host. We currently deliver a meta-heuristic solver being used in the daily operations of the home-care services in a growing number of municipalities in Norway. The problem we solve is naturally more complex than the problem you will solve in this project, but the basic nature of the problem is identical. You can read more about us in a later section in this document.

Timeline

Wednesday, February 14th: Project description is available.
Tuesday, February 27th (10:15 - 12:00): Guest lecture with Visma Resolve @ auditorium F2.
February 14th to March 15th: Work on the project.
Friday, March 15th: Demo day.

The Home-Care Vehicle Routing Problem Variant

Background

Vehicle routing problems (VRPs) are classical combinatorial optimization problems that have received much attention in recent years [3,4,5]. This is due to their computational complexity, wide applicability, and economic importance. VRP formulations are used to model an extremely broad range of issues in many application fields: transportation, supply chain management, production planning, and telecommunication, to name a few. In a large number of practical situations, additional or revised constraints are defined to satisfy real-life scenarios, yielding many different variants of the VRP.

A typical VRP can be stated as follows [3,4,5]:

- A set of geographically dispersed customers with known demands are to be serviced by a (often homogenous) fleet of vehicles with limited capacity.
- Each customer is to be fully serviced exactly once.
- Several vehicles are assigned to a depot.
- A vehicle starts at a depot and has to return to the same depot.
- The objective is to minimize or maximize some goal. An example objective is to minimize the total time spent travelling for all vehicles.

In this project, you will solve a vehicle routing problem with time windows. This can be applied to many logistic cases, but in this project we will adapt it to the routing of home-care nurses to patients. This way, it becomes a simplified version of the problem Visma Resolve solves in the home-care service in Norwegian municipalities.

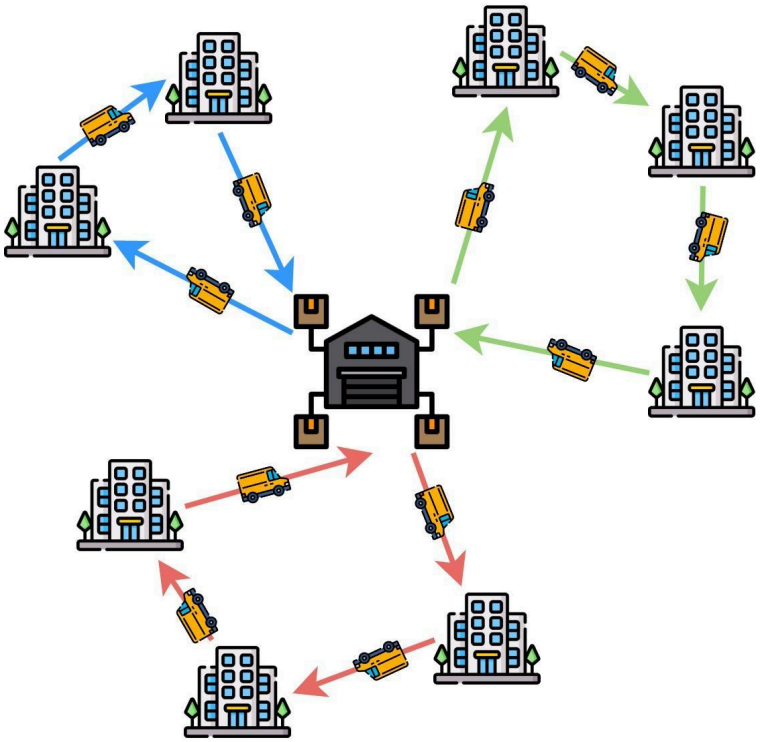

Figure 1: Hypothetical VRP instance with one depot, three vehicles and eight customers.

Problem Description

The following paragraphs describe the problem constraints and objective in detail. These details will be crucial to your implementation of the problem, so please read them carefully.

A set of *nurses* (equivalent to vehicles) cares for a set of *patients* (equivalent to customers). There is one *depot*, and each patient must be visited exactly once. Each nurse starts at the depot, visits an arbitrary subset of the patients to provide care, and returns to the depot. This is called a *route*. Each nurse must be back at the depot before the *return time*.

Each nurse has a *capacity* and each patient has a *demand*. The total demand of the patients in a nurse's route must be less than or equal to their capacity. All nurses in a problem instance have the same capacity, but the patients' demands vary. Practically, the demand can be viewed as the strain on the nurse to perform the patient care. The nurse's capacity is then how much strain a nurse can handle during a route.

Each patient has a *care time* and a *time window*. The care time is the time it takes to care for the patient. The time window has a start and end time. The *start time* defines the earliest time a nurse can start caring for the patient. The *end time* defines the latest time a nurse must be finished with the care. The time windows are strict, meaning that a task cannot begin before the start time and must be completed before the end time. If a nurse arrives at a patient before the start time, they *wait* until the start time before starting the patient care.

The above can be summarized with the following constraints:

1. Each route starts at the depot on time 0.
2. Each route ends at the depot and must arrive before the given depot return time.
3. The total demand on a route must be less than or equal to the nurse's capacity.
4. Each patient visit on a route must be within the respective time windows.
5. Each patient is visited on exactly one route.

The objective is to minimize the *total travel time*, i.e., the sum of the *travel time* of all routes. Note that this *does not* include the care time or the potential waiting time!

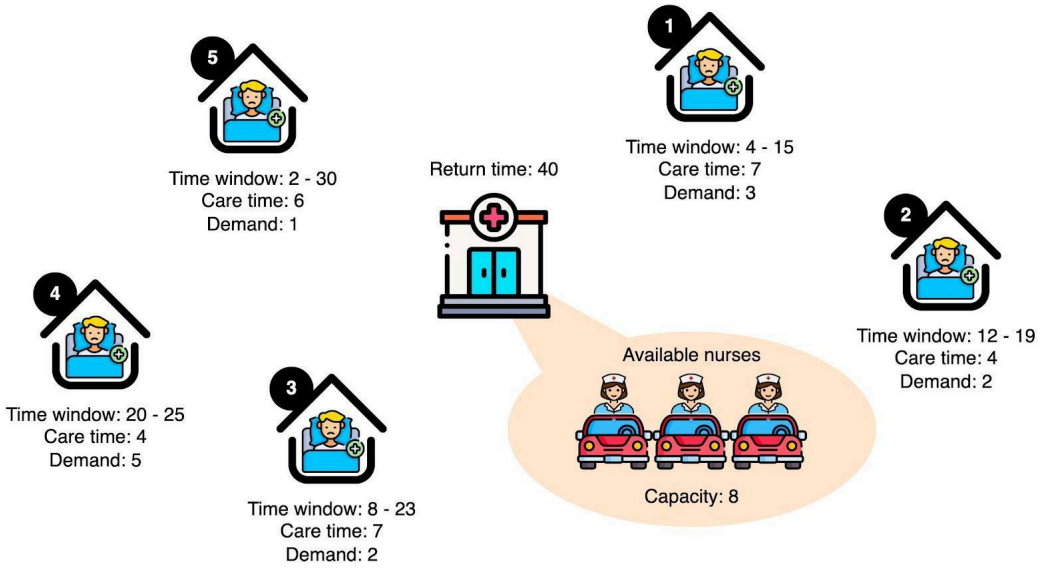

Figure 2: Example of a problem.

Solution Example

A solution to the problem is a route for each nurse. There are many ways to represent a solution. Figure 3 shows a solution example to the problem in Figure 2.

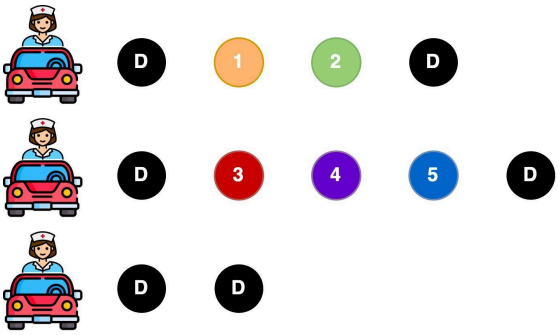

Figure 3: Example of a solution to the problem introduced in Figure 2, not using nurse 3.

This can be translated to a solution representation you can use in your implementation. For example, you could use a list of lists, where each inner list is a route and its elements are the patients that are visited on the route. An empty inner list would represent a nurse not being used. You don't need to include the depot in the inner lists, as all routes start and end there.

Figure 4 breaks down the calculation of a route's patient visit times and duration and how these should be matched with the time windows and return time. The travel and care times are given by the problem instance. Recall that the wait time is only greater than zero if a nurse arrives at a patient before the start time. Note that a route always starts at time 0.

It is important that you remember to distinguish between a route's *duration* and *travel time*. The duration ("Time" in the orange boxes in Figure 4) includes travel, wait, and care time and should be the time that you match with the time windows and return time. The travel time naturally only includes the time spent travelling between patients, and this is the objective value of a solution. Don't forget to include the travel times from and to the depot!

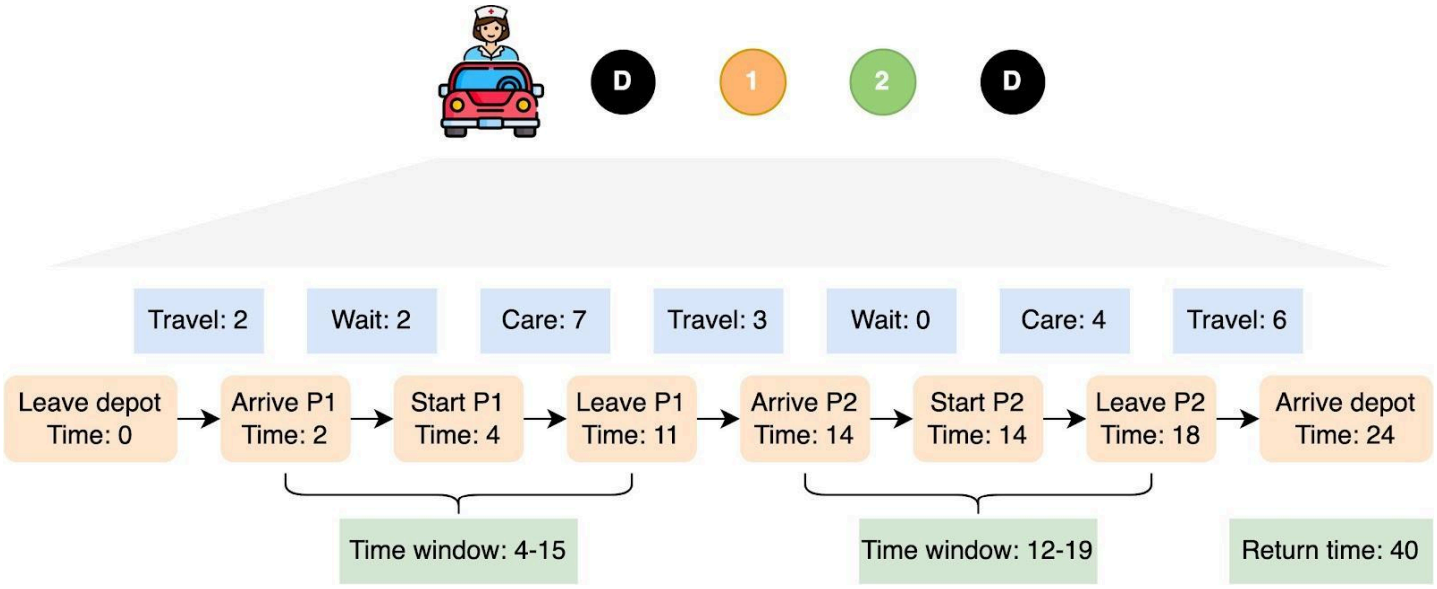

Figure 4: Breakdown of route checkpoints (orange boxes). The travel, wait, and care times are written in the blue boxes, while the green boxes show the time windows and return time.

Description of the Problem Instances

You are given ten instances of the problem outlined in the Problem Description section. The instances are provided as JSON files containing the following fields:

- `instance_name`: The name of the instance.
- `nbr_nurses`: The number of nurses available.
- `capacity_nurse`: The capacity of a nurse.
- `depot`:
  - `return_time`: The time all nurses must be back at the depot.
  - `x_coord`: The x-coordinate of the depot's location (only needed for plot).
  - `y_coord`: The y-coordinate of the depot's location (only needed for plot).
- `patients`: Patient-related information for each patient.
  - `demand`: The patient's demand (strain on nurse performing care).
  - `start_time`: The start time of the patient's time window.
  - `end_time`: The end time of the patient's time window.
  - `care_time`: The time it takes for a nurse to care for the patient.
  - `x_coord`: The x-coordinate of the patient's location (only needed for plot).
  - `y_coord`: The y-coordinate of the patient's location (only needed for plot).
- `travel_times`: The travel time matrix, see details below.

You are advised to test your genetic algorithm implementation on **all** of these instances as you are developing it. The difficulty of the provided instances reflects the difficulty of the unseen instances given on the demo.

The travel time matrix contains travel times between all nodes in the problem instance. This means that it contains the travel times between depot and all the patients, and the travel times from each patient to all other patients. The depot is the first row/column, patient 1 the second row/column, and so on. For example, index (2, 3) and (3, 2) in Figure 5 gives the distance between patient 1 and 2. The travel times are floats, so **do not** round them.

	Depot	Patient 1	Patient 2	Patient 3	Patient 4	Patient 5
Depot	0	2	6	2	7	1
Patient 1	2	0	3	9	2	3
Patient 2	6	3	0	1	5	2
Patient 3	2	9	1	0	7	3
Patient 4	7	2	5	7	0	4
Patient 5	1	3	2	3	4	0

Figure 5: Example of a travel time matrix. Some of the values were used in Figure 2.

Solution Method: Genetic Algorithm

The problem you will solve in this project is NP-hard, which roughly means that an efficient (polynomial time) algorithm for solving all problem instances to optimality is likely unavailable. For this reason, meta-heuristic algorithms are good choices to solve the problem. In this project, you will implement a genetic algorithm (GA) [1,2,3], as you did in Project 1. It might also be beneficial to test whether elitism, local search, and/or parallelization gives better solutions faster, and perhaps also consult the other pointers among the references and lecture materials if needed.

To Do

Mandatory

1. Implement a genetic algorithm that solves the home-care routing problem.
Implement a GA that reads a problem instance and finds increasingly good solutions to that instance. It should have some logging as it runs, and output the best solution found according to the format described in the next section.

It is **not** recommended to implement this algorithm in Python. Python generally runs too slow for successful demonstration during the demo day. Use a faster language like Java, C++, Julia or Rust.

To test your GA during development, we have provided 10 problem instances and *objective benchmarks*. The benchmarks are objective values that should be obtainable, even though they might be tough to reach. They are provided to give you an idea of what “good” solutions are to each instance.

2. Implement code for plotting the solutions.
You must also write code for plotting the solutions of your algorithm. See the next section for details on the plot format. You are free to use any plotting library you want. Python has many easy-to-use plotting tools like matplotlib, plotly, and seaborn. If you use these, you would implement the GA in one of the above-mentioned languages (or a similar, fast one), and do the plotting in Python.

Demo

The demo is divided into two parts: (1) questions about your implementation and theory concepts from the course, and (2) performance tests of your code. You can get a total of 30 points for this assignment, where 6 points are available from the questions and 24 points are available from the performance tests. Everything demo- and score-related will be handled by the course staff and the student assistants, just like Project 1.

Questions (6p)

During the demo, you will be asked questions regarding the behavior and implementation of your code. In these discussions, you may also be asked to explain certain concepts from the theory of this course. The exact questions will not be given in advance, but typical questions could be:

- Make changes to the code to see what effect(s) the change has on the score (the total distance travelled by all vehicles). If there is an improvement in the score, you need to be ready to explain the reason behind this improvement. If there is no improvement, why not?
- Describe and show different parts of the code, such as the chromosome representation, mutation operation(s), and crossover operation(s). Also, be ready to discuss alternatives and the effects of your choices for example with regards to the exploration-exploitation trade-off.
- Explain related theory concepts from the course's curriculum.

Performance Tests (24p)

In this part of the demo you will show us the running code and we will verify that it works. You will be given three **unseen** problem instances that you will run your implemented GA on. You have 30 minutes to finish the test problems (which is the entirety of the demo session). Thus, the run time of the code is important. The questions will be asked as your code is running and finding solutions to the instances.

The point distribution is as follows: Testing three problem instances (24p = 8p x 3). For each problem instance, you can get full or partial scores as follows:

- If your value is within 5% of the benchmark value, or better than the benchmark value, you will get 8 points (full score).
- If your value is within 10% of the benchmark value, you will get 6 points.
- If your value is within 20% of the benchmark value, you will get 4 points.
- If your value is within 30% of the benchmark value, you will get 2 points.
- Otherwise, you will get 0 points.

A plot and a solution text output for the best solution your GA finds must be shown for each of the test instances. The next section describes the expected format of the plot and solution output. You can only get a maximum of 1 point per test if your output is not in the correct format as described (graphically as well as in text format).

Expected Formats

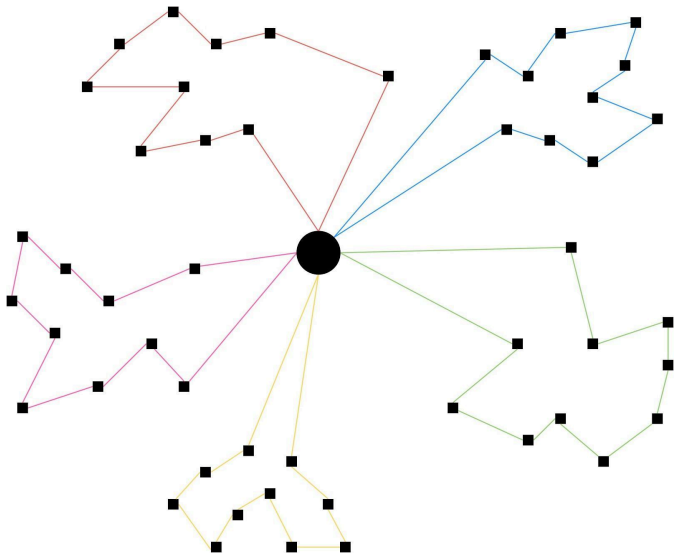

Figure 6: Expected plot format. The most important thing is that you make sure to use different colors for each route. The plot must only be shown for the best-found solution.

```
Nurse capacity

Depot return time
--------------------------------------------------------------------------------
Nurse 1 (N1)      Route duration N1    Covered demand N1    Patient sequence N1
Nurse 2 (N2)      Route duration N2    Covered demand N2    Patient sequence N2
    ⋮                   ⋮                    ⋮                    ⋮
Nurse N (NN)      Route duration NN    Covered demand NN    Patient sequence NN
--------------------------------------------------------------------------------
Objective value (total duration)
```

Figure 7: Expected solution output format. Can be written to the console or saved as a file. The solution output needs only to be shown for the best-found solution.

```
Nurse capacity: 30

Depot return time: 100
--------------------------------------------------------------------------------
Nurse 1    80.63    27    D (0) → 17 (5.14-9.14) [4-15] → ··· → D (80.63)
  ⋮          ⋮      ⋮                      ⋮
--------------------------------------------------------------------------------
Objective value (total duration): 593.17
```

Figure 8: Example of a solution output. Patient 17 is visited (among others, not specified in the example), the visit time is shown within (), and the time window is shown within [].

Delivery Method and Deadline

You should deliver a zip file with your code on BlackBoard. The submission system will be closed on **Friday, March 15th at 09:00 AM**.
The demo day for this project is also **Friday, March 15th**. A signup schedule will be announced on the Friday the week before. Please follow Blackboard for details.

Every student must submit their (jointly) developed code. You must attend the demo individually on the scheduled demo date. No early or late submission or demo will be entertained except in an emergency.

Instances and Benchmarks

Table 1 presents last year's benchmark objective values (travel time) for the training instances found in the zip file on Blackboard. Three similar instances will be provided on the demo day. To get all eight points for an instance, the objective value of the best found solution must be within 5% of the benchmark.

Note that these benchmark values are not necessarily the optimal known values for each problem, but rather provide a benchmark that you might be able to reach. If they are set too strict, the course staff may adjust them after the demo.

Instance	Benchmark	5%	10%	20%	30%
train_0	828	870	911	994	1077
train_1	590	620	649	708	767
train_2	1258	1321	1384	1510	1636
train_3	1133	1190	1247	1360	1473
train_4	1262	1326	1389	1515	1641
train_5	1093	1148	1203	1312	1421
train_6	924	971	1017	1109	1202
train_7	871	915	959	1046	1133
train_8	732	769	806	879	952
train_9	855	898	941	1026	1112

Table 1: Benchmark objective values for all training instances.

About Visma Resolve

In short, Visma is the number one software provider in the Nordics. It is a big organization with more than 14 000 employees and 200 unique software products. It has been growing fast and steady for the previous 20 years, showing no signs of slowing down any time soon.

Resolve is a centralized team in Visma consisting of optimization, machine learning, and infrastructure experts. We love challenging problems, and we love to solve them. We have customers who struggle with some of the most challenging and complex mission-critical problems every single day. Our job is to find those customers, identify their problems and solve them. By dedicating ourselves to the latest research within artificial intelligence, particularly machine learning and optimization, we seek to significantly improve the operations of our customers within healthcare, education and the private sector.

By empowering clients with user-friendly, analytical tools in the cloud, we radically transform their manual processes into competitive advantages, enabling them to make powerful and informed decisions.

We believe that the success of applied optimization and machine learning depends on domain knowledge and a deep understanding of the underlying business. That is why we take part in the whole value chain of the project - from business process improvement to mathematical modelling, algorithm construction, full-stack development, and application hosting. We take pride in being a close-knit group of international, curious and devoted tech talents.

To stay updated on the latest technologies and research, we have frequent knowledge sharing and learning sessions. We also love to arrange hackathons to maintain the team's innovative spirit and drive.

Read more about our team on our website.

References

[1] A. E. Eiben and J. E. Smith. “Introduction to Evolutionary Computing,” 2nd Edition, Springer 2015, pages 67 - 70 (permutation representation) & pages 203 - 211 (constraint handling).

[2] D. Simon. “Evolutionary Optimization Algorithms,” Wiley 2013, pages 449 - 478 (combinatorial optimization incl. TSP).

[3] B. Ombuki-Berman and F. T. Hanshar. “Using Genetic Algorithms for Multi-depot Vehicle Routing.” In: F. B. Pereira and J. Tavares (eds). “Bio-inspired Algorithms for the Vehicle Routing Problem.” Studies in Computational Intelligence, vol 161. Springer 2009. https://doi.org/10.1007/978-3-540-85152-3_4

[4] J. R. Montoya-Torres, J. L. Franco, S. N. Isaza, H. F. Jiménez, N. Herazo-Padilla. “A literature review on the vehicle routing problem with multiple depots.” Computers & Industrial Engineering, Volume 79, 2015, pages 115-129, https://doi.org/10.1016/j.cie.2014.10.029.

[5] K. Braekers, K. Ramaekers, I. V. Nieuwenhuyse. “The vehicle routing problem: State of the art classification and review.” Computers & Industrial Engineering, Volume 99, 2016, pages 300-313, https://doi.org/10.1016/j.cie.2015.12.007.