NATCOR 2024 Practical Challenges

Introduction

We offer four practical challenges, each designed to provide you with a hands-on learning experience. Your team has to choose two challenges to tackle based on the interests and goals of your group.

During the submission process, you will need to decide which of the two challenges is primary and which one is secondary. You are expected to spend more time on your primary challenge during the final presentation. Also, you will have an opportunity to score extra points for your primary challenge.

Brief overview of the challenges

1. **MILP and Complexity**
   - Participants will prove a complexity result about an optimisation problem and develop a solver based on Mixed Integer Linear Programming.
   - Problem domain: Absolute Robust Shortest Path (ARSP).
2. **Heuristics and Meta-heuristics:**
   - Participants will explore heuristic and meta-heuristic approaches for optimisation.
   - Problem domain: TSP.
3. **Automated Configuration (iRace) and Selection Hyper-heuristics (HyFlex):**
   - Participants will utilise automated configuration techniques with iRace and implement selection hyper-heuristics using the HyFlex framework.
   - Problem domain: TSP.
4. **Big Data and Machine Learning for Energy Prediction (Regression):**
   - Participants will work on a machine learning regression problem focused on energy prediction using big data and machine learning techniques.
   - Problem domain: Regression modelling.

Submission

The submission deadline is 9 am on Friday. Please submit your challenges using the following form: `https://forms.office.com/e/597xe6QB09`. For challenges 1, 2 and 4, you will submit Colab notebook links; for challenge 3, you will submit a Github link.

Presentation

The final presentation serves as an opportunity for you to showcase your group work for practical challenges, share your findings, and demonstrate your understanding of the course material.

Each presentation should be up to 8 minutes in length, followed by a brief Q&A session. You are expected to spend around 5 minutes on the primary challenge and around 3 minutes on the secondary challenge. For each of the two challenges, please start by talking about the methodology (the approach taken to address the problem, including any algorithms, techniques, or tools used) and follow up with the results and analysis (the findings and outcomes of the project, including any experimental results, or insights gained). For the primary challenge, also aim to demonstrate your code working (e.g., solving one instance or producing one graph).

Challenge 1: MILP and Complexity

Relevant lectures:

- Complexity Theory (Andrew Parkes; Monday, Tuesday)
- Introduction to Optimisation (Jason Atkin; Monday)

Brief description. This challenge is about the Absolute Robust Shortest Path problem (ARSP). You will prove (on paper) that the problem is NP-hard. You will also produce a mixed-integer linear programming (MILP) formulation of the problem and use it with a commercial MILP solver to build a solution method.

Detailed description

The shortest path problem is to find the shortest path (sequence of nodes) between two given nodes. For example, Google Maps uses the shortest path problem to give directions between two locations.

Formally, you are given a graph $G(V, E)$ with $n = |V|$ nodes. Two of these nodes are dedicated as the origin and destination: $s \in V$ and $t \in V$, respectively. Each edge $(u, v) \in E$ is assigned a weight $w(u, v)$. The objective is to find the shortest path from s the v. The length of a path is the sum of the weights of the edges in that path:

$$f(P) = \sum_{j=1}^{|P|-1} w(P_j, P_{j+1}),$$

where $P = (P_1, P_2, \ldots, P_{|P|})$ is a path.

The shortest path problem is well-known to be polynomially solvable, see Dijkstra's algorithm.

The "Absolute Robust Shortest Path" (ARSP) problem is a generalisation of the shortest path problem. In ARSP, you consider several scenarios; for example, in an internet-connected car navigator, one scenario could be for a light traffic prediction on the M25 near to Heathrow, whereas the other scenario could be for a heavy traffic prediction on that same stretch.

The objective is to find a robust path, that is a path that will work relatively well in all the scenarios. The underlying idea is that the usual "shortest/quickest path" might not be very reliable. Other routes might usually take longer, but are less likely to have a very long delay. This might be relevant if trying to catch a plane – it is important to reduce the chance that some unforeseen traffic conditions lead to missing the flight.

As an example, suppose there are two sensible routes, P and Q. In a typical traffic scenario, P takes 2hr10 and Q takes 2hr20, but in a bad traffic scenario P takes 2hr55 and Q takes 2hr30. In these circumstances, Q might be considered a safer choice.

To define multiple scenarios, we extend the input data of the shortest path problem. Now, the edge weights are defined as $w(u, v, i)$ instead of $w(u, v)$. Here $i \in \{1, 2, \ldots, k\}$ is the scenario index and k is the number of scenarios. We can then calculate the length of a path P in a scenario i:

$$f(P, i) = \sum_{j=1}^{|P|-1} w(P_j, P_{j+1}, i),$$

The objective of ARSP is to find a path that minimises $\max_{i=1}^{k} f(P, i)$.

Task 1.1: Prove that ARSP is NP-hard for the case $k = 2$, that is with just 2 scenarios. Use a reduction FROM the PARTITION problem, which is known to be NP-hard, to the ARSP with 2 scenarios.

Take the PARTITION problem to be defined by a set S of m positive integers $\{a_1, \ldots, a_m\}$. The reduction needs to convert any such problem into some graph with 2 scenarios, and in which the optimal solution of the ARSP can be used to solve the PARTITION decision problem.

Hint: In a 2-scenario ARSP, we denote the costs of each edge by a pair (c_1, c_2) for the costs in scenarios 1 and 2. In the reduction you only need to use

- for each number $a_i \in S$, one edge with cost $(a_i, 0)$ and one edge with cost $(0, a_i)$. The graph should be such that any path from the start to target node must use one of these two edges.
- various other edges with costs $(0, 0)$. These are used to connect the other edges in such a way that an ARSP path creates a partition of S into 2 sets, with the costs in each scenario corresponding to the sums of each of the two partitions.

Warning! This might well be more challenging than the later tasks within this challenge, so you might want to have an initial attempt at this, and otherwise return to it later. If needed, further help and hints will be provided.

You only need to provide the reduction "on paper".

Optionally, you could also write a program that will convert some file-based formats for the instances of the two problems.

Task 1.2: Produce a mixed-integer linear program (MILP) for ARSP. An overview of MILP and some hints can be found in the Colab Notebook.

Task 1.3: Implement a solver for ARSP based on the Gurobi mixed-integer programming solver.

If you have time, compare how it performs on various instances; are there any instances that are 'hard' for your solver, i.e. make it slow? To access the template, please open this Google Colab notebook and save a copy of it: `https://colab.research.google.com/drive/1yKf82YHVbcj8u19OwtqO5rL--kHg3Wiz?usp=sharing`

Warning! If you edit the notebook without saving it to your own Google Drive, you will not be able to save the changes and they will be lost.

You might consider using graph instances that are produced by using the reduction from PARTITION in Task 1.1.

Minimum expectations. To pass this challenge, you are expected to produce

1. A sketch of a reduction for Task 1.1.
2. An (almost complete) MILP formulation of the shortest path problem.

Challenge 2: Heuristic and Metaheuristic Optimisation Algorithms

Relevant lectures:

- Design of Heuristic Algorithms (Dario Landa-Silva, Tuesday)

Brief description. This challenge is to work on some heuristic solution methods for the Travelling Salesman Problem (TSP). You will be given a Python template with several routines already implemented. You will implement some variation operators (also called moves), some metaheuristics, and then analyse their performance.

Detailed description

You are given a template in Python:

1. Instance class that stores the problem instance data;
2. Solution class that includes code for storing a solution (tour), calculating the solution fitness, checking solution feasibility, and visualising the solution;
3. A list of benchmark problem instances to be used in Task 2;
4. The Random initialisation method (the tour is a random permutation of the nodes);
5. The Insert Iterative Improvement (for each node in the tour, try removing it and inserting it in a different position); this is an implementation of the General Iterative Improvement explained in the lecture.
6. An example of creating a problem instance, applying the Insert construction heuristic followed by the Insert hill climber and visualising the obtained solution.

To access the template, please open this Google Colab notebook and save a copy of it: `https://colab.research.google.com/drive/1rJOm9DzdD75iPOiia1caLjzfkk3yvTXi?usp=sharing`
Warning! If you edit the notebook without saving it to your own Google Drive, you will not be able to save the changes and they will be lost.

Task 2.1. Implement the following:

1. The Nearest Neighbourhood initialisation method (start with a random node; keep adding the closest unvisited node until all nodes are visited);
2. The Two-Opt iterative improvement (try removing two edges and restoring the tour by inserting two new edges); this is the 2-opt or 2-edge exchange variation operator explained in the lecture.

3. The Iterated Local Search metaheuristic based on the Two-Opt neighbourhood. You will need to design the perturbation operator yourself. Use time budget as the stopping criterion.

Task 2.2. Analyse the performance of the following methods:

1. The Random vs Nearest Neighbourhood initialisation methods;
2. The Insert Iterative Improvement vs Two-Opt Iterative Improvement;
3. The Iterated Local Search metaheuristic based on Insert vs Two-Opt iterative improvements.

The specific analysis methodology is up to you (some suggestions are given in the lecture). The aim is to compare the above search methods according to the quality of solutions they produce and their running times.

Minimum expectations. To pass this challenge, you are expected to produce

1. An implementation of the Nearest Neighbourhood initialisation method.
2. An implementation of the Two-0pt local search.
3. A comparison of the quality of the solutions obtained by all of the following methods: Random initialisation method, Nearest Neighbourhood initialisation method, Insert Iterative Improvement, and Two-Opt Iterative Improvement.

Challenge 3: Automated Configuration (irace) and Selection Hyper-heuristics (HyFlex)

Relevant lectures:

- Hyper-heuristics (Ender Ozcan, Wednesday)
- Automated Configuration of Optimisation Algorithms (Manuel Lopez-Ibañez, Wednesday)
- Automated Algorithm Design (Rong Qu, Thursday)

Resources are available here.

Brief description You are provided with several example hyper-heuristics with **HyFlex** and example usage of **irace** for automatic configuration. This challenge focuses on enhancing the performance of these hyper-heuristics through automatic algorithm configuration and other techniques.

Detailed description

In this challenge, you will work with example hyper-heuristic algorithms implemented with **HyFlex**. These hyper-heuristics operate on low-level heuristics that have parameters 'depthOfSearch' (DoS) and 'intensityOfSearch' (IoM).

You are provided with

- A set of example hyper-heuristics with **HyFlex**. The provided examples are tailored for solving TSP.
- Example configuration scenario settings for automatic configuration of one example hyper-heuristic with **irace**.
- Instructions for implementing the given examples and a live demo on Wednesday.

Your task is to enhance the performance of these hyper-heuristics using various techniques to obtain better performance for solving TSP.

There might be many ways to improve the results we obtained here. A few ideas for you to think about:

- Automatic configuration for provided hyper-heuristics.
- Implement new high-level strategies.

Minimum expectations. To pass this challenge, you are expected to produce

1. An implementation of the provided example hyper-heuristics and a comparison of their performance on TSP.
2. An implementation of the provided automated configuration example.

Challenge 4: Spark MLlib Challenge - Predicting Appliances Energy Consumption

Relevant lectures:

- Big Data and Machine Learning (Isaac Triguero, Thursday)

Brief description. This challenge is to use Apache Spark's MLlib to construct a machine learning pipeline for a regression problem. Using a dataset from the UCI Machine Learning Repository, you will pre-process the data, train a regression model, and fine-tune it using techniques such as cross-validation. The aim is to create an efficient predictive model while identifying the most relevant input features.

Detailed description

In this challenge, you will delve into the use of machine learning with Apache Spark's MLlib to tackle a regression problem: predicting appliances' energy consumption based on wireless sensor and weather data.

You are given

- a dataset: The provided dataset contains a comprehensive record of energy consumption from appliances over a span of approximately 4.5 months, alongside corresponding temperature and humidity conditions measured by a wireless sensor network. Additionally, weather data from the nearest airport weather station is included. The Colab notebook includes the code for downloading and saving the data.
- a Colab notebook: This contains a guided tutorial for creating a machine learning pipeline with Spark MLlib and evaluating the model.

To access the template, please open this Google Colab notebook and save a copy of it: `https://colab.research.google.com/drive/1lyBlz8shnkqZRhBDH_euHrM47OTA4tRu?usp=sharing`
Warning! If you edit the notebook without saving it to your own Google Drive, you will not be able to save the changes and they will be lost.

The notebook provides a guided tutorial, building on the MLlib example you will see in the lecture to build a pipeline and begin to investigate the data and results. There are several small tasks you will need to fill in the code for. Some parts you can complete based on the lecture example, but for others you will need to investigate the Spark SQL and MLlib documentation yourself!

The challenge is to explore some options beyond this and see if you can improve on the model from the tutorial. This may require doing some brief study of machine learning concepts if you are unfamiliar with this area, and further exploring the functions available to you in the Spark MLlib. There might be many ways to improve the results, but some starting ideas are listed in the second part of the notebook.

Minimum expectations. To pass this challenge, you are expected to:

1. Implement Spark MLlib pipelines to investigate at least one aspect of the data preparation stage, and at least one aspect of the regression model, that could be changed to improve the results.
2. Evaluate and compare your implemented pipelines using at least one suitable evaluation metric.