Majsa Ammouriova, Juliana Castaneda, Rafael David Tordecilla and Angel A. Juan

Heuristics to Solve a Team Orienteering Problem

Abstract

Combinatorial optimization problems are challenging, especially in the real world. Several heuristics could be utilized to solve them. These heuristics differ in their characteristics and solutions found to the problems. One example of the real-world problem was the picking and distribution of face shields at the beginning of the Covid-19 pandemic. Solutions to this problem were needed every day to arrange their picking and distribution. This problem was modelled as a team orienteering problem. Accordingly, heuristics might be used to solve it. These heuristics might be modified to handle instances of large problems and the stochasticity of different data input. This chapter presents the basics of the team orienteering problem and two heuristics used to solve it. Python might be used to realize the heuristics and run experiments to compare the heuristics.

Keywords

Team orienteering problem, heuristics, GRASP, savings-based heuristic, logistics, healthcare logistics

1 Overview

The Internet of Things (IoT) has been a rapidly expanding technology at the industrial level. Nowadays, a network of objects is surrounded by electronic systems, software, sensors, and network connectivity, which enable the collection, storage, and exchange of large data. This network requires optimal data management systems that enable the efficient operation of physical processes. In transportation logistics, more and more systems are being used that automatically monitor vehicle movement, location, status, among other parameters, and generate alerts or make intelligent decisions about them. Solving problems in transportation and logistics is a challenging task today. These problems might be formulated as combinatorial optimization problems. In a combinatorial optimization problem, the search space consists

of a finite space of elements. These elements are selected and arranged in a solution. The number of potential solutions increases exponentially with the number of elements in the search space.

In order to solve combinatorial optimization problems, several heuristics could be utilized. As a result, an optimum or promising solution might be identified depending on the problem and the heuristic. For example, an optimum solution could be guaranteed for small problems. However, for large problems, a promising solution might be identified.

Solving a combinatorial optimization problem becomes challenging if real-time problems are being solved. For example, the integration of the Internet of Things (IoT). In this case, a solution should be recommended within a small time window, immediately. Agile optimization concepts are utilized to handle these problems.

In this chapter, a real case study is presented. This case study appeared at the beginning of the Covid-19 pandemic because of the shortage of face shields. Volunteers printed these shields using their 3D printers, and other volunteers registered to pick up these printed items. This problem is an example of IoT integration and was a challenging task because the decisions were to be made by the following day to assign routes for each volunteer driver. The number of drivers varied from one day to another. To solve this problem, several heuristics might be used. Their performance was evaluated and compared. As a result, students can compare the performance of different heuristics using a real-world case study.

1.1 Didactic Fundamentals

1.1.1 Target Group

The course has been designed for a lecture on analytics for bachelor's or master's students in industrial engineering, statistics, and logistics.

1.1.2 Prerequisites

Basic knowledge of programming, analytical capabilities, applied optimization, data plotting, scientific paper reading, and report writing is required.

1.1.3 Learning Resources

Learning materials involve presentations and videos, reading, and exercises. These materials are uploaded onto a Moodle learning platform. In the following sections, the basics related to this use case are presented.

1.2 Learning Objectives and Competence

Learning outcomes in this chapter based on Bloom's taxonomy and lab-specific psychomotor skill extensions are:

- Remember
  - Students should remember and define the team orienteering problem (TOP).
  - Students should be able to reproduce program codes to solve the TOP.
- Understand
  - Students should distinguish between different methods used to solve the TOP.
  - Students should report their findings in a short report.
- Apply
  - Students should apply the GRASP heuristic learned to solve the TOP's instances.
- Analyze
  - Students should analyze the results obtained and compare the performance of heuristics in different experiments.
- Evaluate
  - Students should evaluate their code and modify it.
- Create
  - Students should be able to describe their results in a report.
- Specific psychomotor skills provided through lab-elements
  - Students will learn how to establish their Python program to solve the TOP.
  - Students will learn how to evaluate a heuristic used to solve the TOP.

2 Use Case

2.1 User Story

Martha is an expert in Operations Research who lives in Barcelona and is worried about the growing number of Covid-19 cases in this region and the increasing burden they are placing on healthcare centers. Her brother, Michael, who is a doctor and works long shifts in a hospital, has told her about shortages of protective elements such as face shields, ear

savers, door openers, and similar sanitary items. Although these elements were not frequently used before the pandemic, the crisis caused healthcare staff to require extra protection, increasing the demand for such elements unexpectedly. Michael and his colleagues in the hospital knew about an initiative called "Coronamakers", or simply "Makers". This initiative is a new community of people who have 3D printers at their respective houses and volunteered to provide these items to hospitals and healthcare centers.

Nevertheless, once the hundreds of volunteers in the surrounding area of Barcelona guaranteed the protective items' production, a logistics problem arose: volunteers were unable to deliver the printed elements by themselves, given both the lockdown restrictions and a large number of items. Hence, only a few external vehicles were able to visit Makers' houses to collect the 3D-printed items and deliver them to healthcare centers. During a conversation about this problem between Martha and Michael, she told him about some math and computational tools she had employed to solve similar problems in the past. She explained to him that, in general, these tools are called heuristics, and they provide fast and good solutions to transportation, manufacturing, and other complex problems. Therefore, heuristics are quite suitable to solve the logistics problem since, although they are not capable of providing optimal solutions, the speedy growth in Covid-19 cases required good-quality solutions that could be obtained in a short time (minutes or seconds).

Immediately after this conversation, Martha and Michael contacted the Makers. Martha offered to solve this logistics problem, together with her research group at Super open University. Moreover, Michael recruited a group of six friends who worked as volunteer drivers. All the friends expressed their willingness to join this project and claimed to be ready once Michael and Martha indicated both the Makers' houses that each driver should visit and the sequence in which these visits had to be carried out. Hence, this was a real and complex problem that Martha's research group had to solve by defining collection routes to maximize the number of items collected.

Martha's group needed to consider a series of conditions or constraints that had to be met to keep the computational model as realistic as possible. Firstly, all the Makers' houses and the healthcare centre locations were identified using Cartesian coordinates. Secondly, the drivers' time was limited, i.e., each route could not exceed a maximum number of hours per day due to curfew hours during the pandemic period. Given both these constraints and the limited number of volunteer drivers, the drivers might not visit all the Makers on the same day. Therefore, an additional decision had to be made regarding which Makers should be visited. Fortunately, since this case represents a daily challenge, Martha knew that Makers who were not

visited on a particular day could be visited on the following days. Thirdly, vehicles were considered virtually unlimited in capacity since the size of the items to be transported was small. Finally, both the travel times between the locations and the visiting time in each house were known. The former is the time taken to travel between any pair of houses or healthcare centres. The latter is the time spent by the driver in carrying out a collection at each house.

After a short discussion with her colleagues at the research group, Martha concluded that this real-world problem should be modelled as the Team Orienteering Problem (TOP). Two reasons led to this decision: the TOP allowed some houses to be skipped, given the strict time limit; and the TOP's typical objective maximized the total reward collected after visiting the houses. In this case, it was so obvious to Martha's group that the reward each Maker offered was the number of 3D-printed items to be collected.

2.2 Tasks

Tasks for students:

- Formulate the TOP to be solved by Martha and her colleagues:
  - Define decision variables
  - Define the objective's function
  - Define the constraints
- Propose more than one heuristic to solve the problem
- Implement the heuristics using Python
- Run experiments and compare the heuristics based on the results obtained

3 Team Orienteering Problem

The TOP is derived from the orienteering problem (OP), an outdoor sport practiced in a mountainous area, where a player has a compass and a map. The player starts at a specific checkpoint from which he/she has to visit as many checkpoints as possible within a set time limit and finally return to the starting point. Each checkpoint has a score associated with it, so the game's objective is to maximize the total score. As the time to return to the starting point is limited, not all checkpoints can be visited. Therefore, the player must select the checkpoints with the highest contribution to his/her total score (Chao et al., 1996). The OP is an NP-hard problem that can be considered a combination of the knapsack problem and the travelling sales-

man problem (TSP) (Vansteenwegen et al. 2011). The knapsack problem is an optimization problem in which a number of items are placed inside a fixed-size knapsack. The items have a given weight, and the objective is to fit as many items as possible into the knapsack given the weight constraint on it (Salkin & De Kluyver, 1975). The TSP is an optimization problem with a given list of cities and distances between each pair of cities, and the objective is to find the shortest possible route that allows the players to visit each city exactly once and return to the origin city (Flood, 1956). When the game is extended from a single individual to teams of two or more players, it is called TOP. Each team member must visit as many selected checkpoints as possible within a given time and, then, return to the starting point. Thus, each checkpoint is visited once, and the total score is maximized (Chao et al. 1996).

The TOP was modelled by Chao et al. (1996) as a multi-level optimization problem. In the first level, the nodes to be visited are selected. In the second level, the selected nodes are assigned to the vehicles in the fleet. Finally, the construction of the routes for each vehicle is done on the third level. According to Gunawan et al. (2016), the TOP can be mathematically defined as a set of nodes $N = [1, \ldots, |N|]$, where each node $i \in N$ is associated with a non-negative reward, r_i. The start node and the end node are described by node 1 and $|N|$, respectively. The objective function of the TOP maximizes the total reward collected from selected nodes by determining routes that are limited by a given time budget, T_{max}, and the time between nodes i and j is t_{ij}. It is assumed that rewards can be added and that each node can be visited once at most.

The problem is formulated as an integer programming model with the following decision variables: $x_{ij} = 1$ if a visit of node i is followed by the visit of node j, otherwise it is 0; and u_i is used in subtour elimination constraints and allows the position of the nodes visited in the route to be determined (cf. Gunawan et al. 2016); subtours represent round tours. The objective's function maximizes the total rewards collected (Equation. 1). The constraints (cf. Gunawan et al. 2016) ensure that: *(i)* routes start from node 1 and end at node $|N|$ (Equation 2); *(ii)* the connectivity of routes guarantees that each node is visited once at most (Equation 3); (*iii*) the total travel time is limited by T_{max} (Equation 4); (*iv*) subtours are prevented (Equations 5 and 6).

$$Maximize \sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} r_i x_{ij} \quad (1)$$

subject to:

$$\sum_{j=2}^{|N|} x_{1j} = \sum_{i=2}^{|N|-1} x_{i|N|} = 1 \quad (2)$$

$$\sum_{i=2}^{|N|-1} x_{ik} = \sum_{j=2}^{|N|} x_{kj} \leq 1; \ \forall k = 2, \ldots . (|N|-1) \quad (3)$$

$$\sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} t_{ij} x_{ij} \leq T_{max} \quad (4)$$

$$2 \leq u_i \leq |N|; \ \forall i = 2, \ldots, |N| \quad (5)$$

$$u_i - u_j + 1 \leq (|N|-1)(1-x_{ij}); \ \forall i = 2, \ldots, |N| \quad (6)$$

4 Heuristic 1: Greedy Randomized Adaptive Search

The greedy randomized adaptive search procedure (GRASP) is the first heuristic described to solve the TOP. It belongs to trajectory methods based on using a single solution and seeing how it evolves or when the number of iterations increases. The basic concepts and key information on Python implementation of GRASP are presented below.

4.1 GRASP Basic Concepts

GRASP is a metaheuristic and a global optimization algorithm. The solution strategy consists of an iterative random sampling of greedy stochastic solutions and the use of a local search heuristic to refine them to a locally optimal solution (Feo & Resende, 1995). Conceptually, GRASP is composed of two phases: (*i*) the intelligent construction of an initial solution through the greedy lexical function, and (*ii*) a local search near the constructed solution to find an improvement. Throughout the process, the best global solution is maintained. Feo & Resende (1995) present the basic generic pseudocode of a generic GRASP as shown in Figure 1. The first two lines correspond to the inputs of the problem. After that, the iterative process occurs between lines 3 and 9. Lines 4 and 5 are the GRASP construction and local search phases, respectively (detailed in Figures 2 and 3). This process is iterative and checks whether the solution generated is better than the best solution found. Accordingly, the best solution is updated as is shown in lines 6 to 8. Finally, the iterative process ends if a stopping criterion is met, such as the maximum number of iterations is reached, and the best solution is returned.

```
Procedure GRASP (MAX_ITERATIONS, SEED)
    1 Best_solution = 0;
    2 Read_Input();
    3 for k=1,2,..., MAX_ITERATIONS Do
    4 Solution = Greedy_Randomized_Construction (SEED);
    5 Solution = Local_Search(Solution);
    6 If Solution is better than Best_solution Then
    7 UpdateSolution(Solution, Best_solution);
    8 end if
    9 end for
    10 return (Best_solution);
end GRASP
```

Figure 1: Pseudocode of a generic GRASP based on Feo & Resende (1995)

Figure 2 presents the pseudocode of the greedy randomized construction, which utilizes uniform randomization to select the most promising elements of a restricted list of candidates. The restricted list of candidates includes solution elements and restricts the characteristics of elements that can be selected in each iteration. The number of selected elements from the lists might be constrained by a pre-specified number (n) or a percentage of the number of elements in the list. A logical type of behaviour is defined to guarantee a random selection of elements from the list to explore a solution space. This list is sorted from the most promising element to the least promising one based on their effect on the objective's function. Each item in the list is assigned a probability (p) of being selected. Then, this list is reduced, considering the n of most promising elements.

Procedure GreedyRandomizedConstruction (SEED)
1 Solution = 0;
2 Sort the candidate elements according to their incremental costs;
3 while solution is not complete **Do**
4 Build the Restricted Candidate List;
5 Select from the Restricted Candidate List and element v at random;
6 Solution = Solution ∪ {v};
7 Re-sort the candidate elements according to their incremental costs;
8 end while
9 return (Solution);
end GreedyRandomizedConstruction

Figure 2: Pseudocode of a generic GRASP construction phase based on Feo & Resende (1995)

Procedure LocalSearch(Solution)
1 while Solution is not locally optimal **Do**
2 Find s' ∈ N such that f(s') ≤ f(Solution);
3 Solution = s';
4 end while
5 return (Solution);
end LocalSearch

Figure 3: Pseudocode of a generic local search phase based on Feo & Resende (1995)

The greedy randomized construction starts by initializing a solution in line 1 of the pseudocode in Figure 2. In the loop between lines 3 and 9, one feasible solution is iteratively constructed by selecting one element from the list at a time. First, the restricted list of candidates is constructed in line 4. Then, a candidate from the list is randomly selected in line 5 and added to the solution in line 6. At each construction iteration, the choice of the next node is determined by sorting nodes in a candidate list with respect to a greedy function. This function measures the reward of selecting each node. The heuristic is adaptive because the rewards associated with each node are updated at each iteration of the construction phase to reflect changes

brought by selecting the previous node. The probabilistic component of selecting the best candidates does not always select the best one because its behavior is entirely random. Finally, the effect of the selected node on the reward is calculated, and the greedy function is adapted in line 7.

Figure 3 presents the pseudocode of the generic local search phase, which is used to improve constructed solutions. The local search algorithm works iteratively by successively replacing the current solution with a better solution in its neighborhood. It terminates when no better solution in the neighborhood is found. Its effectiveness is based on the proper choice of a neighborhood's structure, efficient neighborhood search techniques, and the starting solution. Thus, the neighborhood structure for a given problem relates a solution to the problem to a subset of solutions based on each solution. A solution is then considered locally optimal if there is no better solution in that subset of solutions.

4.2 Key Information for Python Implementation

The implementation of the GRASP heuristic in Python is based on those made by Jason Brownlee[1] and Sain Panyam[2]. Using the well-known instance called berlin52 is recommended[3]. The parameters to be set to find the best solution, for instance, are the maximum number of iterations (outer loop), the maximum number of iterations without improvement, and the greedy factor, the percentage of the elements in the sorted list to be considered by the algorithm.

Implementing the local search in the GRASP might be formulated as a nested loop and generates new solutions based on a stochastic operator. This operator selects non-consecutive edges and swaps them to obtain new connections between the edges, calculating the Euclidean distance between them. Then, it reverses the edges between them to complete the path. The local search keeps track of the best solution and the new solution. Minor modifications are applied to the original solution, and if the new solution with those changes is better than the original or initial solution, the new solution becomes the new best solution (*more details are found on the Moodle platform*).

1 http://www.cleveralgorithms.com/
2 https://www.saipanyam.net/2011/06/clever-algorithms-python.html
3 http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/XML-TSPLIB/instances/

5 Heuristic 2: Savings-Based Heuristic

The second heuristic is based on the concept of the well-known savings heuristic of Clarke and Wright (1964). This savings-based heuristic handles small and large instances of the TOP (Panadero et al., 2020). The basic concepts and key information on the Python[4] implementation of this heuristic are presented below, and the detailed explanation can be checked on Moodle or in the available references.

5.1 Savings-based Heuristic Basic Concepts

The savings-based heuristic starts with a dummy solution from the origin to the destination, where one route per node is considered; each node is connected to the origin and the destination nodes. Since the origin and destination are two different nodes, the connecting arcs are oriented in a specific direction. Basically, a vehicle leaves the origin depot (node 0), visits node i, and then continues its journey to the destination depot (node $i + 1$). Merging arcs occur when precedence constraints are satisfied. Thus, if a route in this dummy solution does not satisfy the driving range constraint, the associated customer is discarded from the problem; this node cannot be reached with the current fleet of vehicles. Since the objective function of the TOP maximizes the rewards collected in a limited available time, the "saving" is associated with each arc connecting two different customers. This saving is related to the reward obtained by visiting both customers on the same route rather than using two different routes (Panadero et al. 2020). Figure 4 presents the basic generic pseudocode of the savings-based heuristic.

4 https://docs.python.org/3/library/index.html

```
Savings-Based Heuristic (SEED, Nodes)
1 sol  createDummySolution(Nodes);
2 SavingsList  computeSortedSavingsList(Nodes);
3 While (SavingsList is not empty) do
4 edge  selectNextEdge(SavingsLists)
5 iRoute  getStartingRoute(edge)
6 jRoute  getClosingRoute(edge)
7 travelTimeNewRoute  validate MergeDrivingConstraints(NewRoute)
8 isMergeValid  validateMergeDrivingConstraints
9 if (isMergeValid) then
10 sol UpdateSolution(newRoute, iRoute, jRoute, sol)
11 end if
12 deleteEdgeFromList(edge);
13 end While
14 SortRouteByProfit(sol)
15 deleteRoutesByProfit (sol, maxVehicles)
16 return sol
```

Figure 4: Pseudocode of a generic savings-based heuristic according to Juan et al. (2020)

The enriched savings heuristic considers a linear combination of classical savings defined by Clarke and Wright (1964) and the reward associated with an arc. Both quantities must be in the same order of magnitude for this linear combination. Mathematically, this represents the efficiency or enriched savings and defines the relevance given to each of the parameters to be optimized, the original savings based on distance or time, and the utility or reward of the visiting node.

There are two associated savings for each arc, depending on the actual direction in which the arc is traversed. After calculating all the savings, the list of arcs can be sorted from the highest to the lowest savings. From this list, the route merging starts. In each iteration, the arc at the top of the sorted list is selected. This arc connects two routes, which are merged into a new route as long as this new route does not violate the driving range restriction. Finally, the list of routes is sorted according to the total rewards provided to select as many routes as possible from this list, considering the restricted number of vehicles in the fleet (Panadero et al. 2020).

5.2 Key Information for Python Implementation

The implementation of the savings-based heuristic in Python starts by defining different classes: *Node*, *Edge*, *Route*, and *Solution*. The class *Node* contains the information associated with the nodes. The class *Edge* contains the

information on the edges connecting nodes and the concept of efficiency. The class *Route* is a list of connected edges, the path's cost, and the total rewards (demand) collected. The class *Solution* counts the number of solutions and stores related information on each one of them. Detailed information on each class is available in the learning materials available to the students, as is the complete development of the metaheuristic.

For the construction of edges and nodes, a list of nodes is constructed in which the first node is $node[0]$ and the destination node is $node[-1]$. Then, an edge (arc) is created between each node and other nodes, and the Euclidean distance is calculated accordingly. The efficiency list is a linear combination using α and $(1 - \alpha)$, where α is a prespecified factor e.g., 0.7, as explained in Section 5.1. The resulting efficiency list is then sorted from the highest to the lowest value.

The construction of a dummy solution, which is the route from the origin to a node and then to the destination, is used to create the initial solution. The total reward for each route created in the dummy solution is also calculated. From there, the algorithm starts performing the iterative process of edge selection and route merging, based on the following conditions: *(i)* the resulting merged route must not exist in the defined routes; *(ii)* the first node must be linked to the origin and the last one to the destination; and *(iii)* the cost after merging the routes should not exceed the maximum time allowed. The merging process must start with a while loop, where the conditions are evaluated, and the merging process occurs. The loop must be executed several times.

In a final step, the solution is sorted by the merged routes generated. Since the final solution needs a certain number of routes given by the number of vehicles in the fleet, unnecessarily stored information must be eliminated. Finally, printing and plotting the solution using the networkx library is recommended.

6 Further Input: Comparison between Heuristics

In our studies, we would like to compare heuristics and select a heuristic to be adapted. For the comparison, statistical tests are used to differentiate between different approaches and heuristics (Beiranvand et al. 2017). In order to compare heuristics, benchmark data sets are used. The heuristics are compared with respect to their performance, such as the quality of the recommended solutions and efficiency (Beiranvand et al. 2017). The difference between recommended solutions and the best-known solution

defines the quality of the solutions, and the efficiency of a heuristic might be represented by the time required to get a solution.

In the statistical tests statements, we define hypotheses (Montgomery and Runger 2007, Sheskin 2011). A null hypothesis, H_0, states that no difference between the heuristics exists, while an alternative hypothesis, H_1, defines a claim to be tested. For example, $H_0: S_1 - S_2 = 0$ and $H_1: S_2 > S_1$. H_0states that solutions S_1 and S_2 found by two heuristics do not differ, and H_1 states that the solution found by heuristic 2 is better than the solution found by heuristic 1 with respect to profit, $S_2 > S_1$. The statistical tests are performed with strong evidence required to reject H_0. If H_0 is rejected, H_1 is accepted.

The selection of a statistical test depends on the parameter being tested, S, and the sample size. In our comparisons, the sample size is defined by the number of heuristic runs on a benchmark data set. Examples of statistical tests are parametric tests and non-parametric tests (Sheskin 2011). Several assumptions are required to utilize parametric tests: (*i*) tested samples are selected randomly from their populations; (*ii*) the distribution of the population follows normal distribution; and (*iii*) the variance of the population is homogenous. If one of the assumptions is violated, non-parametric tests should be used. Non-parametric tests have the advantage of being suitable to test small samples with ($n < 30$).

If a heuristic is run more than 30 times on a benchmark data set, a *t*-test could be used (Montgomery and Runger 2007). In the *t*-test, the mean of the solutions, μ, of the two heuristics considered is calculated and compared. The *p*-value of the test defines "the smallest level of significance that would lead to the rejection of the null hypothesis H_0 with the given data" (Montgomery and Runger 2007, p. 300). The *p*-value is compared to α, significance level. If the *p*-value is smaller than α, H_1 is accepted and it is concluded that both heuristics differ significantly ($\mu_1 \neq \mu_2$). Otherwise, H_0 cannot be rejected. The common values for α are 10% or 5%.

In addition to statistical tests, we can plot the experiment's results and tabulate them to highlight the difference between heuristics (Beiranvand et al. 2017). For example, the change of the best solutions found by a heuristic could be plotted versus the number of iterations. The best solutions found by a heuristic in different runs can be tabulated and compared to other heuristics by calculating the difference between them, a gap.

7 Assessment

Students work in groups to solve the routing problem, like the one described in Section 2 with their respective tasks (Section 2.2), implementing the heuristics in Python and comparing the heuristics described. First, students should select a problem to solve. Then, they design their experiments and run them. Finally, the results collected should be tabulated and analyzed in a group report. In advanced challenges, students modify the heuristics to become more agile by introducing biased randomization (the material is available on Moodle).

The group report is submitted to a tutor for feedback on the analysis and the experiments performed. Students use this feedback to assess their work and benefit from it in their future analyses. In addition, students can have group discussions to discuss their findings and ideas.

Abbreviations

GRASP	Greedy Randomized Adaptive Search Procedure
IoT	Internet of Things
OP	Orienteering Problem
TOP	Team Orienteering Problem
TSP	Traveling Salesman Problem

References

Beiranvand, V., Hare, W., & Lucet, Y. (2017). Best practices for comparing optimization algorithms. Optimization and Engineering, 18(4), 815–848. https://doi.org/10.1007/s11081-017-9366-1.

Chao, I. M., Golden, B. L., & Wasil, E. A. (1996). The team orienteering problem. European journal of operational research, 88(3), 464–474. https://doi.org/10.1016/0377-2217(94)00289-4.

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. Operations research, 12(4), 568–581. https://doi.org/10.1287/opre.12.4.568.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. Journal of global optimization, 6(2), 109–133. https://doi.org/10.1007/BF01096763.

Flood, M. M. (1956). The traveling-salesman problem. *Operations research*, *4*(1), 61–75. https://doi.org/10.1287/opre.4.1.61

Gunawan, A., Lau, H. C., & Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research, 255(2), 315–332. https://doi.org/10.1016/j.ejor.2016.04.059.

Juan, A. A., Freixes, A., Panadero, J., Serrat, C., & Estrada-Moreno, A. (2020). Routing drones in smart cities: A biased-randomized algorithm for solving the team orienteering problem in real time. *Transportation Research Procedia*, *47*, 243–250. https://doi.org/10.1016/j.trpro.2020.03.095

Montgomery, D. C., & Runger, G. C. (2018). Applied Statistics and Probability for Engineers (7th ed.). Wiley.

Panadero, J., Juan, A. A., Bayliss, C., & Currie, C. (2020). Maximising reward from a team of surveillance drones: A simheuristic approach to the stochastic team orienteering problem. European Journal of Industrial Engineering, 14(4), 485–516. https://doi.org/10.1504/EJIE.2020.108581.

Salkin, H. M., & De Kluyver, C. A. (1975). The knapsack problem: a survey. *Naval Research Logistics Quarterly*, *22*(1), 127–144. https://doi.org/10.1002/nav.3800220110

Sheskin, D. (2011). Handbook of parametric and nonparametric statistical procedures (5th ed.). CRC Press.

Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: A survey. European Journal of Operational Research, 209(1), 1–10. https://doi.org/10.1016/j.ejor.2010.03.045.

1 Template Didactical Concept — Handout for Teachers

Title Name of the Concept

Heuristics to Solve the Team Orienteering Problem

Lab Environment

X-Heuristics in Intelligent Transportation, Sustainable Logistics, and Smart Cities.

2 Didactical Analysis

Students should utilize Python to optimize a given function. Then, they are asked to construct their Python code for the greedy randomized adaptive search procedure (GRASP) and the Clarke & Wright Savings (CWS) heuristic to solve the team orienteering problem (TOP).

Target Group

The course targets bachelor's and master's students in industrial engineering, statistics, and logistics. These students need basic programming knowledge, analytics, applied optimization and simulation, data plotting, scientific paper reading, and report writing. The requirements target beginners, while optional exercises target advanced or expert students.

Institutional Requirements

The primary resource is a computer and access to the material. A Python language environment should be installed, e.g., Pycharm, to run experiments. The tutor needs to be familiar with the problem presented and the Python code used to solve it. These problems are fundamental in transportation and logistics, and the Python programming language is one of the easiest programming languages. Students may raise questions regarding running experiments and debugging the code.

Learning Objectives

- Students should remember and define the TOP.
- Students should be able to reproduce program codes to solve the TOP.
- Students should understand the basic heuristic used to solve the TOP.
- Students should apply the heuristic learned to solve the TOP's instances.

- Students should analyze the results obtained.
- Students should evaluate their code and modify it.
- Students should be able to describe their results in a report.

Subject Matter

This handout is related to the "Heuristics to Solve the Team Orienteering Problem" educational chapter. In this chapter, the GRASP and Clarke and Wright heuristics are introduced. In addition, implementation of Python is presented. The explanations are presented as videos in the Moodle course, and additional reading is recommended.

3 Didactical Concept

Methodical Implementation

- The concept of chapters is presented in a video, where the problem and the heuristic to solve it are presented.
- The students are asked to be divided into groups with 2 to 3 students per group to work on the exercises; thus, the learning is collaborative.
- A bigger group discussion could be arranged for all students in some circumstances.

Media

The material required is explained in videos (uploaded on the Moodle course). In the next version of the course, quizzes could be added there.

Learning Organization

Students work in groups; thus, students interact with their colleagues in their group. Further discussion could be arranged between groups, especially for optional exercises. These students could use the video conferencing platforms for meetings and discussions. The students could contact the tutor if they do not find an answer to their question or need more help. The videos are about one hour long, and students require discussions after the video and to do experiments; thus, the lecture's content and exercises are scheduled to last two weeks.

Feedback and Evaluation

Students should present their experiment results as plots/reports and explain their findings. Thus, the evaluation could be based on their analysis and understanding of the presented concept. Each chapter is evaluated after two weeks of its release, and the evaluation may include suggestions to improve the analysis. This feedback enhances students' learning and their analysis.

Expert Tips

Similarly to the situation in any programming lab, students raise many basic questions regarding code implementation. Students should be advised to use the debug to understand the code and rectify their errors.

Authors

Dr. Majsa Ammouriova
Universitat Oberta de Catalunya
Internet Interdisciplinary Institute
Rambla del Poblenou, 156
08018 Barcelona, Spain
mammouriova@uoc.edu

Eng. Juliana Castaneda
Universitat Oberta de Catalunya
Internet Interdisciplinary Institute
Rambla del Poblenou, 156
08018 Barcelona, Spain
jcastanedaji@uoc.edu

Eng. Rafael David Tordecilla
Universitat Oberta de Catalunya
Internet Interdisciplinary Institute
Rambla del Poblenou, 156
08018 Barcelona, Spain
rtordecilla@uoc.edu

Prof. Angel A. Juan
Universitat Oberta de Catalunya
Internet Interdisciplinary Institute
Rambla del Poblenou, 156
08018 Barcelona, Spain
https://ajuanp.wordpress.com/
ajuanp@uoc.edu