**Design and Test for Vehicle Routing Algorithms Constrained**

**by Energy Recharge Transactions**

Project Step 2

4 February 2022

**Let’s recall the 2nd step:**

* Program a heuristic that builds a solution from an instance,
* Check the correctness of the code by testing the heuristic on several instances: use the function written in step 1 to check automatically that the solution is valid, then check by hand, on small instances, that the solution built by the heuristic is the expected solution.

# How’s the heuristic working?

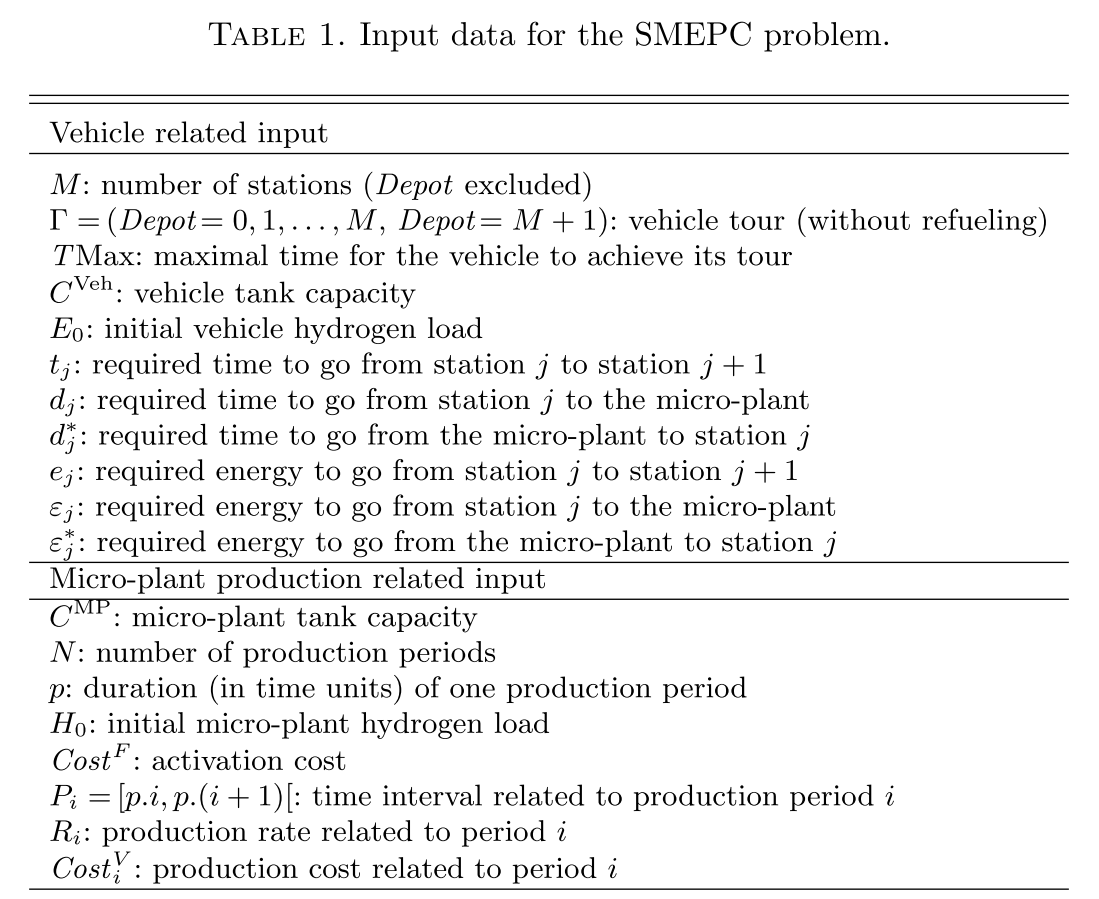
First let’s recall that our problem contains 2 sub-problems:

* the *refuelling* sub-problem: after which stations will the vehicle refuel?
* the *production* sub-problem: during which periods does the micro-plant produce?

Our heuristic will be driven by the first sub-problem: we are going to build the best refuelling schedule for the vehicle assuming that the micro-plant always has enough hydrogen in stock (*heuristic step 1*). Then, we will adapt the production schedule to satisfy the refuelling schedule decided in the heuristic step 1 (*heuristic step 2*).

# Heuristic step 1 (*refuelling sub problem*)

We use the same notations as [1]:



We note the additional distance covered by the vehicle to refuel after station :

We need to add the time to refuel.

## A modelling for the refuelling problem

To achieve the heuristic step 1 we construct a directed edge-weighted graph whose set of vertices are the stations (including the depot) and whose arcs represent the possible routes for the vehicle to refuel. There are 3 types of arcs:

* **type 1 arcs:**

This arc represents the route

“station micro-plant station station micro-plant” (see Figure 1)

Its cost is .



Figure 1. Route represented by a type 1 arc

* **type 2 arcs:**

This arc represents the route

“depot station micro-plant”

Its cost is .

* **type 3 arcs:**

This arc represents the route

“station micro-plant station final depot”

Its cost is .

We can merge the arcs of type 1 and 3 by setting :

* **type 1 and 3 arcs:** :

Its cost is .

The best solution of the *refuelling* sub-problem is then given by a shortest path in the graph .

## Example

Let’s consider a problem with 4 stations, . The energy and distance costs are as follow:



We can compute the detours:

;

The graph G, constructed from this example, is (the value on each arc represents its cost):



The shortest path is of cost .



The meaning is: the vehicle refuels after station 0 and after station 3 and the cost is increased by 3 time units compared to a direct path (without refuelling).

## Algorithm: how to compute the shortest path?

In fact, the shortest path can be computed without explicitly implementing the graph .

We use a *label-setting algorithm*: each vertex of is associated with a label which gives both the cost of a shortest path from 0 to and the parent vertex which is the predecessor of in this shortest path. This algorithm is similar to the well-known *Dijkstra algorithm* but adapted to our particular graph.

**Implementation phase**

A first algorithm (Algorithm 1) computes the labels for all vertices.

A *label* is a pair **(pred, cost)**. We use an array **T** of labels to store the labels of each vertex .

At the end of the algorithm **T[j].pred** contains the predecessor of in a shortest from 0 path to and **T[j].cost** contains its cost.

// Labels Initialization

T[0] := (-1,0);

**for** j = 1 to M+1 do T[j] := (-1, ); **endFor**

//compute the labels related to type 2 arcs

j := 1;

**while** (j <= M **AND**) **do**

T[j] := (0,0);

j := j+1;

**endWhile**

//compute the labels related to type 1 and 3 arcs

**for** i = 0 to M **do**

costCur := T[i].cost;

j := i+1;

**while** (j <= M+1 **AND** ) **do** //there is an arc from i to j

**if** (T[j].cost > T[i].cost + detour(i)) **then**

T[j] := (i, T[i].cost + detour(i)); //update the shortest path to j

**endIf**

j := j+1;

**endWhile**

**endFor**

**Algorithm 1**

**Question:** If we run Algorithm 1 on the previous example, what labels do we get?

We now want to retrieve a shortest path from to using the labels computed by Algorithm 1 in order to fill the array refuelSt (see project step 1 for the definition of refuelSt). This is done by the Algorithm 2.

T[M+1] provides us with the cost of a shortest path from to . We start from this label and go back to vertex following the predecessors given by the labels:

**Initialize** refuelSt to false;

cur := T[M+1].pred;

//follow the predecessor labels

**do**

refuelSt[cur] := true;

sav := cur;

cur := T[cur].pred;

**while** (cur != 0) **endDo**;

//there is a particular case with the depot (see remark below)

**if** **then**

refuelSt[0] := true;

**endIf**

**Algorithm 2**

**Remark**: Since there is 2 types of arcs whose origin is 0 (with 2 different meanings regarding to the refuelling schedule), we need to find out which one has been used in the shortest path: if it possible to use a type 2 arc then we used this one (because its cost is 0) otherwise we used a type 3 arc.

refuelSt[0] becomes true only in the case that we used a type 3 arc.

## Improving the modelling

In the previous modelling (section 2.1) the node of the auxiliary graph refers to the initial depot and an arc with origin can have 2 different meanings: “the vehicle refuels after the depot” or “the vehicle does not refuel after the depot” depending on its type. This can be confusing.

We propose here a new modelling (very close to the previous one) in which the node is distinct from the node depot. We still have 3 arc types but their meaning is totally defined by their origin: an arc whose origin is means “the vehicle refuels after station ”(the vehicle refuels after the depot in case ).

In the following subsections, we explain the main ingredients of this new modelling.

### The graph

The graph is now as follow:

Its set of vertices is the set . and represent the depot at the beginning and at the end of the vehicle journey respectively. A node corresponds to the fact that the vehicle is at the micro-plant after it has left station , or just after leaving the depot if .

**Type 1 arcs** : An arc indicates that the vehicle is able to travel from the micro-plant (after station ) to the micro-plant (after station ) without any refuelling operation. It exists if

**Type 2 arcs**: An arc , means that:

* the vehicle can go from the depot to the micro-plant just after the depot (if ) ,
* the vehicle can go from the depot to station *j* without any refuelling (if ).

The initial energy must be sufficient for this trip. Therefore, it exists if

**Type 3 arcs**: Arcs mean that the vehicle can go from the micro-plant after station , to the final depot without refuelling. Moreover, the vehicle can arrive at the depot with at least its initial energy .

### The weights on the arcs

Let be a path between and in . The length of is equal to

As in2 , we denote by the additional distance (or extra time) covered by the vehicle to refuel after station :

Thus let us assign the following weight or cost to each arc of .

* **Type 1 arcs and type 3 arcs**: Their cost is .
* **type 2 arcs:** its cost is arbitrarily taken as .

As in 2.1, the arcs of type 1 and 3 can be merged by setting . In short, we have for type 1 and 3 arcs :

,

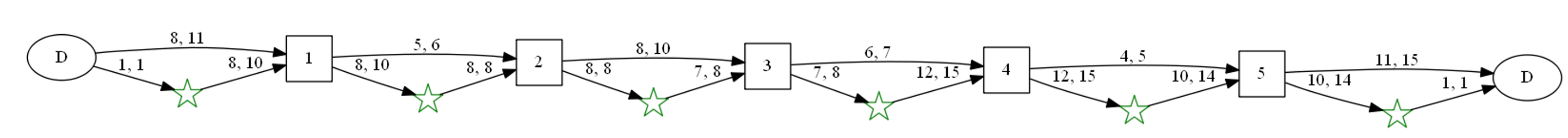
.

### The refuelling problem

The refuelling problem consists in determining the shortest path in the graph .

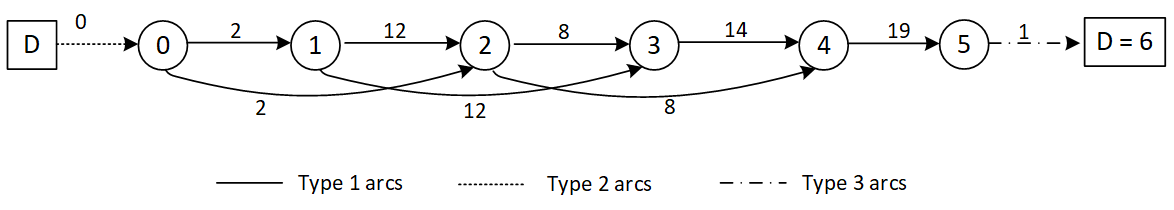
### Example

Let us consider the following instance (instance “Jean 2”):



The costs on the arcs are (time, energy). The initial hydrogen load in the vehicle tank () is 20 units and its capacity () is 30 units.

The graph for the modelling of the refuelling problem is as follows:



The shortest path is: D -> 0 -> 2 -> 4 -> 5 -> 6 of cost 30 and it means that the vehicle refuels after depot and after stations 2, 4 and 5.

For each refuelling, we can compute the minimal quantity of hydrogen for the vehicle to refuel in order to be able to go to the next refuelling (or final depot). It comes that:

* **first refuel**: The vehicle arrives at the micro-plant with 19 () hydrogen units in its tank and it needs 24 to go to the next refuel. So it needs to refuel 5 units.
* **second refuel**: The vehicle arrives at the micro-plant with 0 hydrogen units in its tank and it needs 30 to go to the next refuel. So it requires to refuel 30 units.
* **third refuel**: The vehicle arrives at the micro-plant with 0 hydrogen units in its tank and it needs 28 to go to the next refuel. So it requires to refuel 28 units.
* **fourth refuel:** The vehicle arrives at the micro-plant with 0 hydrogen units in its tank and it needs 21 to arrive at the final depot with hydrogen units in its tank. It requires to refuel 21 units.

Thus, we compute the total amount of hydrogen needed by all the vehicle refuellings.

# Heuristic step 2 (*production sub problem*)

In the heuristic step 2 we need to compute a production schedule, that is to say, we need to decide during which periods the micro-plant produces.

To do so, we already know the total amount of hydrogen needed by all the vehicle refuellings. Now, we make the micro-plant produce until at least hydrogen units have been generated.

Several constraints must been respected:

* the micro-plant cannot produce when the vehicle refuels,
* the micro-plant tank capacity must never be exceeded,
* the vehicle can refuel only if the amount of hydrogen in the micro-plant tank is sufficient: the vehicle wait until this amount is reached.

**Questions**

1. Give the production schedule for the previous example.

2. Design an algorithm to compute such a schedule from any refuelling schedule.

# References

[1] Fatiha Bendali, Eloise Mole Kamga, Jean Mailfert, Alain Quilliot and Hélène Toussaint. **Synchronizing energy production and vehicle routing**. RAIRO-Oper. Res. Volume 55, Number 4, July-August 2021.