1 Introduction

In this research, we deal with the problem related to the synchronization between energy production with a hydrogen micro-plant and its consumption, by a fleet of autonomous vehicles in charge of logistic tasks inside a closed area. We call this problem *Synchronazed Vehicle Routing and Energy Production* (SVREP) problem. On the one hand, the production manager schedules the activity of the micro-plant, taking into account that both production costs and production capacities are time dependant. On the other hand, the fleet schedules and routes the vehicles in such a way they efficiently achieve a set of internal logistic tasks. Both meet in order to perform refuelling transactions, when the vehicle moves toward the micro-plant in order to refuel. As it is a decision problem too complex, we deal with the vehicle routing and refuelling using a surrogate formulation of the cost related to production. We try two kind of such surrogate formulations: the first one is based upon the fact that, for a fixed routing tour Γ, an optimal refueling decision can be obtained in polynomial time by solving a simple parametric auxiliary problem. The second formulation relies on an approximation by a neural network of the cost of the global problem when fixing the tour Γ and the refuelling transactions.

2 ILP formulation of SVREP

In this section, we formulate the SVREP as an Integer Linear Programming(ILP) model. In the Table 1, the input of the problem is shown.

Table 1: Input data for the SVREP.

Vehicle related input
M: number of stations (*Depot* excluded)
J: the set of stations. 0 and M+1 are the depot and -1 is the micro-plant.
TMax: maximal time for the vehicle to achieve its tour
C^{Veh}: vehicle tank capacity
E_0: initial vehicle hydrogen load
$T_{j,k}$: required time to go from station j to station k
$E_{j,k}$: required energy to go from station j to station k
α: conversion ratio of time into cost
Micro-plant production related input
C^{MP}:micro-plant tank capacity
N: number of production periods
p: duration (in time units) of one production period
H_0: initial micro-plant hydrogen load
$Cost^F$: activation cost
$P_i = [p \cdot i, p \cdot (i+1)]$: time interval related to production period i
R_i:production rate related to period i
$Cost_i^V$:production cost related to period i

The SVREP is clearly too complex to be properly handle through ILP machinery. Still, it

is interesting for us to set such a model, since it will be useful for benchmarking, and also to derive from this model the surrogate model relative to the vehicle routing and refueling. The variables of the model are:

- **Production variables:**

- $z_i \in \{0,1\}$ with $i = 0, ..., N-1 : z_i = 1$ means that the micro-plant is active during the period i;
- $y_i \in \{0,1\}$ with $i = 0, ..., N-1 : y_i = 1$ means that the micro-plant is activated at time $p \cdot i$;
- $V_i^{Prod} \geq 0$ with $i = 0, ..., N : V_i^{Prod}$ = hydrogen load of the micro-plant tank at the start i.
- $\delta_i \in \{0,1\}$ with $i = 0, ..., N-1 : \delta_i = 1$ the vehicle refuels during the period i;
- $L_i^* \geq 0$ with $i = 0, ..., N-1 : L_i^*$ = quantity of hydrogen loaded by the vehicle during period i.

- **Vehicle variables:**

- $Z_{j,k} \in \{0,1\}$ with $j, k = -1, ..., M+1 : Z_{j,k} = 1$ iff the vehicles moves from j to k;
- $X_{j,k} \in \{0,1\}$ with $j, k = 0, ..., M+1 : X$ describes the route followed by the vehicle without the refuelling transactions;
- $L_j \geq 0$ with $j = 0, ..., M : L_j$ = quantity of hydrogen loaded by the vehicle after station j;
- $\tau_j \geq 0$ with $j = 0, ..., M+1 : \tau_j$ = time when the vehicle arrives at station j;
- $\tau_j^* \geq 0$ with $j = 0, ..., M+1$: if $Z_{j,-1} = 1$, then τ_j^* = time when the vehicle starts refuelling after station j;
- $V_j^{Veh} \geq 0$ with $j = 0, ..., M+1 : V_j^{Veh}$ = hydrogen load of the vehicle tank when the vehicle arrives at j.

-**Synchronization variables:** $U_{i,j} \in \{0,1\}$ with $i = 0, ..., N-1, j = 0, ..., M : U_{i,j}$ means that the vehicle is going to refuel during period i after traveling from j to the micro-plant.

Then, the objective function comes as follows:

- **Objective function:** Minimize $\sum_{i=0,...,N-1}(Cost^F \cdot y_i + Cost_i^V \cdot z_i) + \alpha \cdot \tau_{M+1}$.

We must now explicit the constraints, while distinguishing those related to production, to the routing of the vehicle and to the synchronization between both processes.

- **Production constraints:**

- For any $i = 1, ..., N-1 : y_i \geq z_i - z_{i-1};\ y_0 \geq z_0;$ [E1]
- For any $i = 0, ..., N-1 : z_i + \delta_i \leq 1;$ [E2]
- $V_0^{Prod} = H_0;\ V_n^{Prod} \geq H_0;$ [E3]
- For any $i = 0, ..., N-1 : V_{i+1}^{Prod} = V_i^{Prod} + z_i R_i - L_i^*;$ [E4]
- For any $i = 0, ..., N-1 : L_i^* \leq V_i^{Prod} \leq C^{MP}$ and $L_i^* \leq \delta_i \cdot C^{Veh}$. [E5]

Explanation: The constraint [E1] expresses that the micro-plant is activated in period i if it wasn't active in $i-1$ and it becomes active in period i, considering that before the first period it was inactive. The constraint [E2] imposes that the producing and refueling simultaneously is forbidden. [E3] expresses the initial and final conditions of energy load at the micro-plant. The constraint [E4] expresses the evolution of the micro-plant's load and [E5] that the refueling load cannot exceed neither the capacity of the micro-plant nor the vehicle, and it has to be 0 if there isn't a transaction.

- **Vehicle constraints:**

- $Z_{m+1,0} = 1$; For any $j : Z_{j,j} = 0$; [F1]
- For any $j = 0, ..., M+1 : \sum_{k=-1,...,M+1} Z_{j,k} = 1 = \sum_{k=-1,...,M+1} Z_{k,j}$; [F2]
- $\sum_{j=0,...,M+1} Z_{j,-1} = \sum_{j=0,...,M+1} Z_{-1,j} \geq 1$; [F2Bis]
- $\forall j, k, X_{j,k} \geq Z_{j,k}$; [F3]
- For any $j = 0, ..., M+1 : \sum_{k=0,...,M+1} X_{j,k} = 1 = \sum_{k=0,...,M+1} X_{k,j}$; [F3Bis]
- $V_0 = E_0$; $V_{M+1} \geq E_0$; [F4]
- For any $j = 1, ..., M : E_{j,-1} \leq V_j \leq C^{Veh}$; [F4Bis]
- For any $j, k = 0, ..., M+1 : Z_{j,k} + (V_k^{Veh} - V_j^{Veh} + E_{j,k})/C^{Veh} \leq 1$; [F5]
- For any $j, k = 0, ..., M : (X_{j,k} - Z_{j,k}) + (V_k^{Veh} - V_j^{Veh} + E_{j,-1} + E_{-1,k} - L_j)/C^{Veh} \leq 1$; [F6]
- For any $j, = 0, ..., M : L_j \leq C^{Veh} + E_{j,-1} - V_j^{Veh}$; [F7]
- For any $j, = 0, ..., M : L_j \leq \sum_{k=0,...,M+1} (X_{j,k} - Z_{j,k}) \cdot C^{Veh}$; [F7Bis]
- $\tau_0 = 0$; $\tau_{M+1} \leq TMax$; [F8]
- For any $j = 0, ..., M, k = 1, ..., M+1 : Z_{j,k} + (\tau_j + T_{j,k} - \tau_k)/TMax \leq 1$; [F9]
- For any $j = 0, ..., M, k = 1, ..., M+1 : (X_{j,k} - Z_{j,k}) + (\tau_j^* + p + T_{-1,k} - \tau_k)/TMax \leq 1$; [F10]
- For any $j = 0, ..., M, k = 1, ..., M+1 : (X_{j,k} - Z_{j,k}) + (\tau_j + T_{j,-1} - \tau_j^*)/TMax \leq 1$; [F10Bis]

Explanation: [F1,F2, F2Bis] mean that the vehicle follows a circular route which visits exactly once every station j and that it refuels at least once. [F3, F3Bis] mean that vector X describes the route followed by the vehicle when bypassing the micro-plant. Notice that if $X_{j,k} = 1$ and $Z_{j,k} = 0$ then it implies that $Z_{j,-1} = 1 = Z_{-1,k}$. [F4] expresses the initial and final conditions of the vehicle's load and [F4Bis] the fact that the vehicle must can go to the micro-plant from every station. [F5] expresses the evolution of the vehicle's loan when traveling from j to k without refueling, and [F6] when it refuels. [F7, F7Bis] mean that the vehicle's load cannot exceed the capacity after refueling and the refueling load is 0 when there isn't a refueling transaction. [F8] expresses the initial and final conditions of the time, [F8] the time evolution when there isn't a refueling transaction and [F10, F10Bis] when there is.

- **Synchronization constraints:**

- For any $j = 0, ..., M : \sum_{i=0,...,N-1} U_{i,j} = Z_{j,-1}$; [G1]
- For any $i = 0, ..., N-1 : \delta_i = \sum_{j=0,...,M} U_{i,j}$; [G2]
- For any $j = 0, ..., M : \tau_j^* = \sum_{i=0,...,N-1} p \cdot i \cdot U_{i,j}$; [G3]

- For any $i = 0, ..., N-1, j = 0, ..., M : U_{i,j} + (L_j - L_i^*)/C^{Veh} \leq 1$; [G4]

Explanation: [G1, G2] mean that the refueling transactions induce a matching between refueling periods and stations. [G3] fixes the time when a refueling transaction after station j starts. The constraints [G4] expresses that the received load L_j doesn't exceed the transferred load L_i^*.

This ILP formulation may be enhance by additional constraints (Cuts). The capacity of the micro-plant imposes at least $(\sum_{i=1,...,N} R_i z_i)/C^{Prod} - 1$ energy transfers from the micro-plant to the vehicle, so it needs at least $(\sum_{i=0,...,N-1} R_i z_i)/C^{Prod}$ activations of the production.

- $\sum_{i=0,...,N-1} y_i \geq (\sum_{i=0,...,N-1} R_i z_i)/C^{Prod}$ [E6]

Also, we may get a lower bound for time value τ_{M+1} as the time to traverse the path without counting the waiting time to refuel:

- $\tau_{M+1} \geq \sum_{j,k=-1,...,M+1} T_{j,k} \cdot Z_{j,k}$ [F11]

Finally, we may notice that, if the vehicle spends an amount W of energy while moving inside or at the boarder of some station subset A, then it must move at least W/C^{Veh} times toward the micro-plant in order to refuel. If we also take into account that the the initial load has to be E_0 and the final load $C^{Veh} - E_0$, it leads to set the following *Extended No Subtour* constraints:

- For any subset A of {0,1,...,M+1}: $\sum_{j \notin A, k \in A} \Pi_{j,k} \cdot Z_{j,k} \geq \sum_{j,k \in A} E_{j,k} \cdot Z_{j,k} + \sum_{j \notin A, k \in A} (E_{j,k} \cdot Z_{j,k} + E_{k,j} \cdot Z_{k,j})$ [F12]
- For any subset A of {0,1,...,M+1}: $\sum_{j \notin A, k \in A} \Pi^*_{j,k} \cdot Z_{j,k} \geq \sum_{j,k \in A} E_{j,k} \cdot Z_{j,k} + \sum_{j \notin A, k \in A} (E_{j,k} \cdot Z_{j,k} + E_{k,j} \cdot Z_{k,j})$ [F12Bis]

Where $\Pi_{j,k} = E_0, \Pi^*_{j,k} = C^{Veh} - E_0$ when $j = M+1, k = 0$, otherwise $\Pi_{j,k} = \Pi^*_{j,k} = C^{Veh}$. Notice that [F12] contains the standard *no subtour* constraint for TSP as soon as $E_{j,k} = 0$ implies $j = k$.

3 Fixed tour refuelling problem

In this section, we explain the refuelling problem when the tour Γ is fixed. This problem can be solved in polynomial time using the parameters α and β, which are the conversion ratios of time into cost and energy into cost, respectively.

3.1 The graph

To solve the refuelling sub-problem we construct a directed edge-weighted graph $G = (V, E)$ whose set of vertices $V = \{D, 0, 1, \ldots, M, M+1\}$ are the stations, where $j = D$ is the depot (0 in the ILP model) and $j = 0$ a fictitious node which represents that the vehicle has to refuel just after the depot. The arcs $(i, j) \in E$ represent the possible routes for the vehicle to refuel. There are 3 types of arcs:

- **Type 1 arcs :** An arc $(j, j') \in E,\ 0 \leq j < j' \leq M$ indicates that the vehicle is able to travel from the micro-plant (after station j) to the micro-plant (after station j') without any refuelling operation. It exists if

$$E_{-1,j+1} + \sum_{k=j+1}^{j'-1} E_{k,k+1} + E_{j',-1} \leq C^{Veh}$$

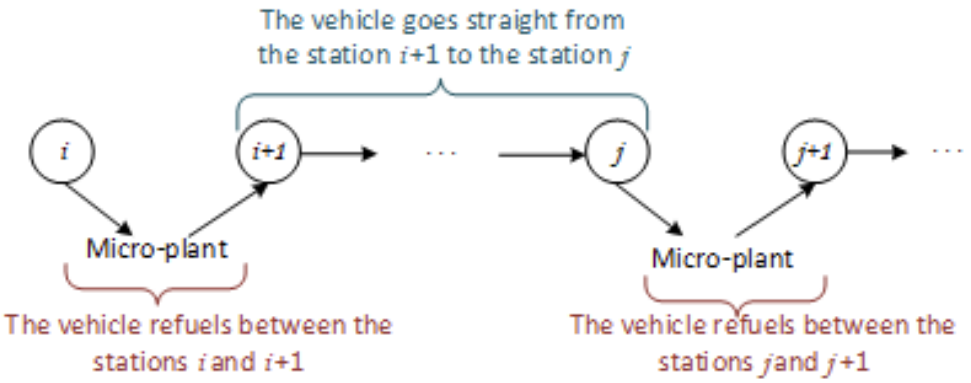


Figure 1: Route represented by a type 1 arc (i,j)

- **Type 2 arcs:** An arc $(D, j) \in E,\ 0 \leq j \leq M$, means that the vehicle can go from the depot to the micro-plant just after the depot (if $j = 0$) or the vehicle can go from the depot to station j without any refuelling (if $j \geq 1$). The initial energy E_0 must be sufficient for this trip, so it exists if

$$\sum_{k=0}^{j-1} E_{k,k+1} + E_{j,-1} \leq E_0$$

- **Type 3 arcs:** Arcs $(j, M+1) \in E, 0 \leq j \leq M$ mean that the vehicle can go from the micro-plant after station j, to the final depot without refuelling. Moreover, the vehicle can arrive at the depot with at least its initial energy E_0.

$$E_{-1,j+1} + \sum_{k=j+1}^{M} E_{k,k+1} + E_0 \leq C^{Veh}$$

We can merge the arcs of type 1 and 3 by setting $E_{M+1,-1} = E_0$:
- **Type 1 and 3 arcs:** $\forall i \in \{0, ..., M\}, \forall j \in \{i+1, ..., M+1\}, (i, j) \in E$ if

$$E_{-1,i+1} + \sum_{k=i+1}^{j} E_{k,k+1} + E_{j,-1} \leq C^{Veh}$$

3.2 The costs on the arcs

The weight of an arc (i, j) is a weighted sum of the time and energy consumed to arrive at the micro-plant after station j. Let us assign the following weight or cost to each arc i, j of G, using the parameters α, β.

- **Type 2 arcs:** $\sum_{k=0}^{j-1}(\alpha T_{k,k+1} + \beta E_{k,k+1}) + \alpha T_{j,-1} + \beta E_{j,-1}$
- **Type 1 arcs an type 3 arcs:** $\alpha \cdot (p + T_{-1,i+1} + T_{j,-1}) + \beta \cdot (E_{-1,i+1} + E_{j,-1}) + \sum_{k=i+1}^{j-1}(\alpha T_{k,k+1} + \beta E_{k,k+1})$

The total cost *cost* of a path $P = (0, j_1, ..j_q, M + 1)$ can be calculates as:

$$costT = \sum_{k=0}^{j_1-1} T_{k,k+1} + T_{j_1,-1} + T_{-1,j_1+1} + p + \sum_{k=j_1+1}^{j_2-1} T_{k,k+1} + T_{j_2,-1} + T_{-1,j_2+1} + p + ...$$

$$+ \sum_{k=j_{q-1}+1}^{j_q-1} T_{k,k+1} + T_{j_q,-1} + T_{-1,j_q+1} + p + \sum_{j_1+1}^{M} T_{k,k+1} =$$

$$= \sum_{k=0}^{M} T_{k+1} + T_{j_1,-1} + T^*_{-1,j_1+1} + p - T_{j_1,j-1+1} + ... + T_{j_q,-1} + T^*_{-1,j_q+1} + p - T_{j_q,j_q+1}$$

$$costE = \sum_{k=0}^{j_1-1} E_{k,k+1} + E_{j_1,-1} + E^*_{-1,j_1+1} + ... + \sum_{j_1+1}^{M} E_{k,k+1} =$$

$$= \sum_{k=0}^{M} E_{k,k+1} + E_{-1,j_1} + E_{j_1+1,-1} - E_{j_1,j_1+1} + ... + E_{j_q,-1} + E^*_{-1,j_q+1} - E_{j_q,j_q+1}$$

$$cost = \alpha \cdot costT + \beta \cdot costE$$

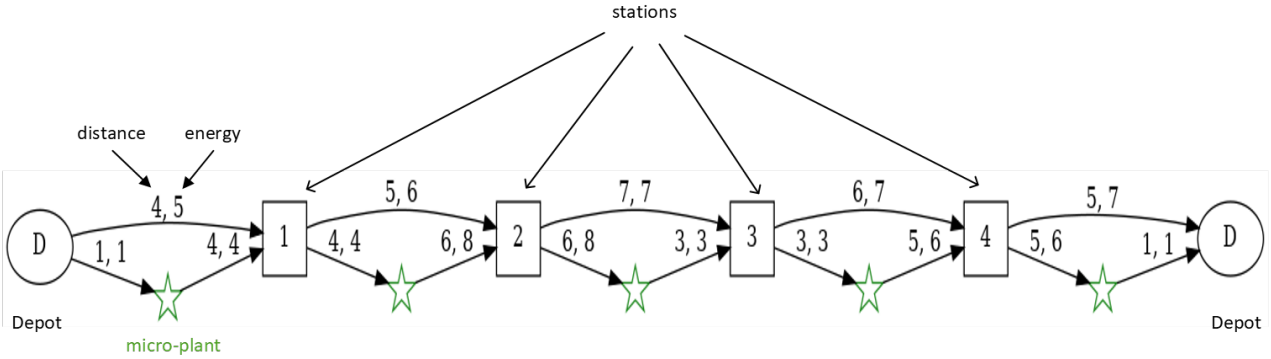
All the paths has the element $\alpha \sum_{k=0}^{M} T_{k,k+1} + \beta \sum_{k=0}^{M} E_{k,k+1}$, so the costs of the arcs can be simplify using the additional cost to refuel after station j, denoted by *detour(j)*:

$$detour(j) = \alpha(T_{j,-1} + T_{-1,j+1} + p - T_{j,j+1}) + \beta(E_{j,-1} + E^*_{-1,j+1} - E_{j,j+1})$$

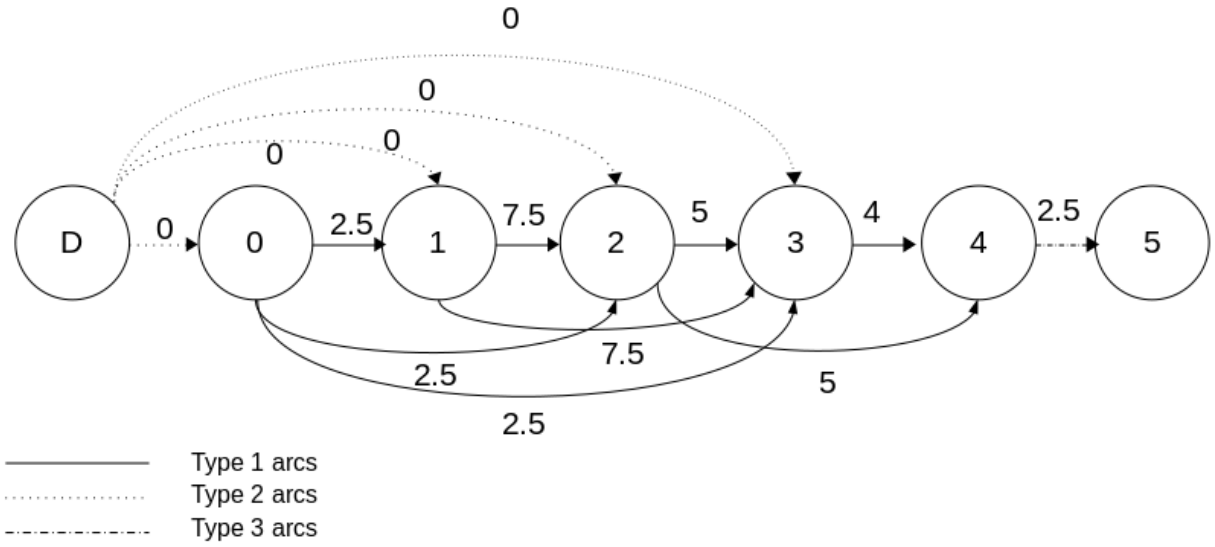
Doing this, the cost of an arc j of type 1 or 3 is the *detour(j)* and the cost of an arc of type 2 is arbitrarily taken as 0.

3.3 Example

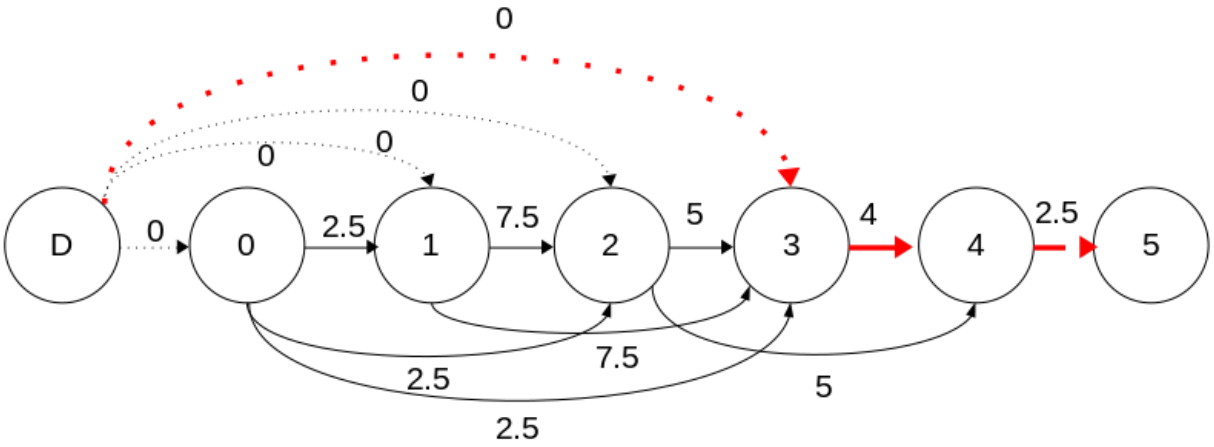
Let's consider a problem with 4 stations, E_0=21, C^{Veh}=24 and p=4. The energy and distance costs are as follow:



The graph G, using $\alpha = 0.5$ and $\beta = 0.5$, is (the value on each arc represents its cost):



The shortest path is $(D, 3) \to (3, 4) \to (4, 5)$ of cost $0 + 4 + 2.5 = 6.5$. The meaning is: the vehicle refuels after station 3 and after station 4. To calculate the cost of the path, we have to add $\alpha \sum_{k=0}^{4} T_{k,k+1} + \beta \sum_{k=0}^{4} E_{k,k+1}$, which gives 36. We have to add this term because the total cost is the weighted sum of the consumed time and energy during the tour.



3.4 Shortest path algorithm

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

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

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

We now want to retrieve a shortest path from D to M+1 using the labels computed by Algorithm 1 in order to fill the array *refuelSt*. This is done by the Algorithm 2. T[M+1] provides us with the cost of a shortest path from 0 to M+1. We start from this label and go back to vertex D following the predecessors given by the labels.

This method that solves the refueling sub-problem will henceforth be referred as $SP^{\alpha\beta}$.

4 Surrogate approach for SVREP

The above ILP method will have difficulties to solve even small instances. For this motive, we are going to focus in solving the routing problem, obtaining a tour Γ and the refuelling transactions. With them, it is possible to solve the production problem for even large instances.

4.1 ILP formulation refuelling problem

A first approach is to get the optimal tour solving an ILP model. As in the case of the fixed tour, we need a conversion ratio β which converts energy into cost, to approximate the cost due to the production cost in the global ILP model. From the previous model, we can get the following ILP surrogate model for only the routing problem:

- Constraints (F1, F2, F2Bis, F4, F4Bis, F5, F12 and F12Bis) be satisfied.

Algorithm 1: Compute the labels for all vertices.

```
/* Labels Initialization                                              */
T[D] ← (-1,0);
for j=0 to M+1 do T[j] ← (-1,+∞);
/* compute the labels related to type 2 arcs                          */
j←0; E← 0;
while (j ≤ M) ∧ (E + E_{j,-1} ≤ E_0) do
    T[j] ← (D, 0);
    E ← E+E_{j,j+1};
    j ← j+1;
end
/* compute the labels related to type 1 and 3 arcs                    */
for i=0 to M do
    j←i+1; E← E_{-1,j}; while (j ≤ M + 1) ∧ (E + E_{j,-1} ≤ C^{Veh}) do
        C ← T[i].cost + detour(j);
        if C < T[j].cost then T[j] ← (i, C);
        E ← E + E_{j,j+1};
        j ← j+1;
    end
end
```

- For any $j = 0, ..., M + 1 : Z_{-1,j} + (V_j^{Veh} - C^{Veh} + E_{-1,j})/C^{Veh} \leq 1$ [F13]
- Objective function: Minimize $\sum_{j,k} Z_{j,k}(\alpha \cdot T_{j,k} + \beta \cdot E_{j,k})$

In this case, the variables of the model are only Z and V. With the contraints [F12, F12Bis], the problem can be solved without the variable V and the constraints [F4, F4Bis, F5 and F13]. However, they are kept because they speed up the program.

An important fact is how the constraints [F12, F12Bis] are managed. The number of subsets increases exponentially with the number of stations. In order to avoid adding to much constraints and make the model slow, we will add the subtour constraints during the execution of the ILP, adding only the subtour constraints that are violated by the current matrix Z. For

Algorithm 2: Retrieve the shortest path from the depot to M+1.

```
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≠D;
```

doing this, we solve a problem similar to the Flow Max/Min Cut problem, that finds if there is a subset A which doesn't satisfy these constraitns. The pseudocode is in the Algorithm 4. The *Separate* algorithm is called inside a callback of CPLEX, and we add the constraint [F12] for the subsets in A^* and the constraint [F12Bis] for the ones in A^{**}.

4.2 Heuristic algorithm

The second approach is to use an heuristic algorithm, which uses the $SP^{\alpha\beta}$ to get an approximation of the global cost for a tour Γ. This algorithm does a local search trying to find a tour which minimizes that approximation.

Let's define the variables of this problem for our heuristic algorithm, which are similar to Z and X of the ILP model, but in this case they will be vectors instead of matrices:
- Tour Γ: A list that encodes the order of the stations in the tour. $\Gamma_j \in \{0, 1, ..., M\}$ with $j = 0, ..., M+1 : \Gamma_j$ is the j^{th} station to visit in the tour, where $\Gamma_0 = 0$ and $\Gamma_{M+1} = 0$ (the tour starts and finishes in the depot).
- A set of the refueling transitions *refuelSt*: We use the $SP^{\alpha\beta}$ algorithm to get the refueling transitions for the tour Γ.

For computing the refueling transitions of a tour Γ using $SP^{\alpha\beta}$, we use the order of stations determined by Γ instead the order $0, 1, ..., M, M+1$.

With the tour Γ and the refueling transitions *refuelSt*, we are going to decompose the tour in sub-tours that start and finish in the micro-plant, except the first which starts in the depot and the second which finishes in the depot. We associate to each sub-tour γ_i the needed time and energy to do it and the index of the last station of the sub-tour. We also save the index of the last sub-tour of γ. The Algorithm 5 makes the decomposition of the tour. From the decomposition, the tour can be obtained adding the stations of $\gamma[0]$, then the stations from $\gamma[2]$ to $\gamma[\text{KAct}]$ and finally the stations of $\gamma[1]$, without adding the micro-plants of the sub-tours.

Now, we will implement two methods to remove and insert elements in the decomposition. The procedure to remove elements receives as input a list R of pairs (k, j), where j is the station to remove from the sub-tour k. As output, it returns the decomposition without the stations $j \in R$ and it updates the values of the vectors EN, TM and SAct associated to the decomposition. For each element in R, the procedure subtracts from EN$[k]$ and TM$[k]$ the energy and time to go from the previous station x to j and the energy and time to go from j to the next station y. Then, it adds the time and energy to go from x to y. Next, the stations after j are shifted and the SAct$[k]$ is reduced by 1.

The procedure to insert an element receives as input the station j, the sub-tour k and the position p where the element has to be inserted. In this case, to update the values of EN$[k]$ and TM$[k]$, the procedure subtracts the time and distance between the station x in the position $p-1$ and the actual station y in p, and it adds the energy and time to go from x to j and from j to y. To add j in the sub-tour, first the stations from y until the last station are shifted to leave space to the new station, and then j is put in the position p.

Algorithm 3: Separation algorithm.

Algorithm *Separate(Z)*:

$\Lambda \leftarrow \sum_{j,k} Z_{j,k} \cdot E_{j,k}$;
(A,ϵ) $\leftarrow$ MinCut($Z, \Lambda, 1$);
if ϵ=*True* **then** Success $\leftarrow$ False; Insert subset A into collection A^*;
else
  (A,ϵ) $\leftarrow$ MinCut($Z, \Lambda, 2$);
  if ϵ=*True* **then** Success $\leftarrow$ False; Insert subset A into collection A^{**};
  else Success $\leftarrow$ True;
return Success;

Function *MinCut(Z, Λ, ξ)*:

H $\leftarrow$ Null vector, with indexation in the arcs (i,j) such that $Z_{i,j} \neq 0$;
H* $\leftarrow$ Null vector, with indexation in the arcs (i,j) such that $Z_{i,j} \neq 0$;
V $\leftarrow$ 0; Stop $\leftarrow$ False;
while $\neg$ *Stop* $\wedge$ $V < \Lambda$ **do**
  if $\xi = 1$ **then** (Γ, δ,A) $\leftarrow$ AugmentingPath1(Z,H,H*);
  else (Γ, δ,A) $\leftarrow$ AugmentingPath2(Z,H,H*);
  if Γ=*Nil* **then** Stop $\leftarrow$ True;
  else
    V $\leftarrow$ V+δ;
    for *(j,k,ε)* **in** Γ **do**
      if $\varepsilon = 2$ **then** $H^*_{j,k} \leftarrow H^*_{j,k} + \delta$;
      If $\varepsilon = 1$ then $H_{j,k} \leftarrow H_{j,k} + \delta$ else $H_{j,k} \leftarrow H_{j,k} - \delta$;
if *Stop* **then return** (A, True);
else return (Nil, False);

Function *Retrieve(Pred,ξ)*:

jCurr $\leftarrow$ M+2; $\Gamma \leftarrow$ Nil; $\delta \leftarrow +\infty$;
if $\xi = 1$ **then**
  while *jCurr* $\neq$ *-1* **do**
    (j,k,ε, β) $\leftarrow$ Pred[jCurr]; $\delta \leftarrow$ Min(δ, β);
    Insert (j,k,ε) as the head of Γ;
    if $\varepsilon > 0$ **then** jCurr $\leftarrow$ j;
    else jCurr $\leftarrow$ k;
else
  while *jCurr* $\neq$ *-1* **do**
    (j,k,ε, β) $\leftarrow$ Pred[jCurr]; $\delta \leftarrow$ Min(δ, β);
    Insert (j,k,ε) as the head of Γ;
    if $\varepsilon > 0$ **then** jCurr $\leftarrow$ k;
    else jCurr $\leftarrow$ j;
return (Γ, δ);

Algorithm 4: Augmenting path functions.

Function *AugmentingPath1(Z,H,H*)*:

L $\leftarrow \{-1\}$; Stop $\leftarrow$ False; Mark[-1] $\leftarrow$ 1;
for *j=1* **to** *M+2* **do** Mark[j] $\leftarrow$ False;
while $\neg$ *Stop* $\wedge$ $L \neq$ *Nil* **do**
  j $\leftarrow$ Head(L); Pop(L);
  if $j \neq$ *M+1* **then**
    Search for some j such that $(H*_{j,k} < Z_{j,k} \cdot E_{j,k})$;
    if *k exists* **then** Pred[M+2] $\leftarrow (j, k, 2, Z_{j,k} \cdot E_{j,k} - H*_{j,k})$; Stop $\leftarrow$ True;
    for *any k such that* $H_{j,k} < Z_{j,k}(C^{Veh} - E_{j,k}) \wedge \neg Mark[k]$ **do**
      Mark[k] $\leftarrow$ True; Pred[k] $\leftarrow$ (j,k,1,$Z_{j,k}(C^{Veh} - E_{j,k}) - H_{j,k}$); Insert(k,L);
    for *any k such that* $H_{k,j} > 0 \wedge \neg Mark[k]$ **do**
      Mark[k] $\leftarrow$ True; Pred[k] $\leftarrow$ (k,j,-1,$H_{k,j}$); Insert(k,L);
  else
    /* Notice that $Z_{M+1,0} = 1$ always. */
    if $\neg Mark[0] \wedge H_{j,0} < E_0$ **then**
      Mark[0] $\leftarrow$ True; Pred[0] $\leftarrow$ (j,0,1,$E_0 - H_{j,0}$); Insert(0,L);
    for *any k such that* $H_{k,j} > 0 \wedge \neg Mark[k]$ **do**
      Mark[k] $\leftarrow$ True; Pred[k] $\leftarrow$ (k,j,-1,$H_{k,j}$); Insert(k,L);
if $\neg$ *Stop* **then** A $\leftarrow$ {j such that Mark[j]=0}; **return** (Nil,0,A);
else $(\Gamma, \delta) \leftarrow$Retrieve(Pred,1); **return** $(\Gamma, \delta,$ Nil);

Function *AugmentingPath2(Z,H,H*)*:

L $\leftarrow \{-1\}$; Stop $\leftarrow$ False; Mark[-1] $\leftarrow$ 1;
for *j=1* **to** *M+2* **do** Mark[j] $\leftarrow$ False;
while $\neg$ *Stop* $\wedge$ $L \neq$ *Nil* **do**
  j $\leftarrow$ Head(L); Pop(L);
  if $j \neq$ *0* **then**
    Search for some j such that $(H*_{k,j} < Z_{k,j} \cdot E_{k,j})$;
    if *k exists* **then** Pred[M+2] $\leftarrow (k, j, 2, Z_{k,j} \cdot E_{k,j} - H*_{k,j})$; Stop $\leftarrow$ True;
    for *any k such that* $H_{k,j} < Z_{k,j}(C^{Veh} - E_{k,j}) \wedge \neg Mark[k]$ **do**
      Mark[k] $\leftarrow$ True; Pred[k] $\leftarrow$ (k,j,1,$Z_{k,j}(C^{Veh} - E_{k,j}) - H_{k,j}$); Insert(k,L);
    for *any k such that* $H_{j,k} > 0 \wedge \neg Mark[k]$ **do**
      Mark[k] $\leftarrow$ True; Pred[k] $\leftarrow$ (j,k,-1,$H_{j,k}$); Insert(k,L);
  else
    /* Notice that $Z_{M+1,0} = 1$ always. */
    if $\neg Mark[M+1] \wedge H_{M+1,0} < C^{Veh} - E_0$ **then**
      Mark[M+1] $\leftarrow$ True; Pred[M+1] $\leftarrow$ (M+1,0,1,$C^{Veh} - E_0 - H_{j,0}$);
      Insert(M+1,L);
    for *any k such that* $H_{j,k} > 0 \wedge \neg Mark[k]$ **do**
      Mark[k] $\leftarrow$ True; Pred[k] $\leftarrow$ (j,k,-1,$H_{j,k}$); Insert(k,L);
if $\neg$ *Stop* **then** A $\leftarrow$ {j such that Mark[j]=0}; **return** (Nil,0,A);
else $(\Gamma, \delta) \leftarrow$Retrieve(Pred,2); **return** $(\Gamma, \delta,$ Nil);

Algorithm 5: Compute the decomposition of the tour.

Input: Γ,refuelSt
Output: γ,EN,TM,SAct,KAct
Initialize EN and TM to 0 ;
k $\leftarrow$1, i $\leftarrow$ 0 ;
for *j=0* **to** *M* **do**
  if *refuelSt[j]* **then**
    γ[k][i] $\leftarrow \Gamma_j$; i $\leftarrow$ i+1;
    `/* Add time and energy from the last station to the micro-plant */`
    TM[k] $\leftarrow$ TM[k]+$T_{\Gamma j,M+1}$;
    EN[k] $\leftarrow$ EN[k]+$E_{\Gamma j,M+1}$;
    γ[k][i] $\leftarrow$ M+1; SAct[k] $\leftarrow$ i;
    k $\leftarrow$k+1;
    `/* Add time and energy from the micro-plant to the next station for the next tour */`
    TM[k] $\leftarrow T_{M+1,\Gamma j+1}$; EN[k] $\leftarrow$ EN[k]+$E_{M+1,\Gamma j+1}$;
    γ[k][0] $\leftarrow$ M+1; i $\leftarrow$ 1;
  else
    TM[k] $\leftarrow$ TM[k]+$T_{\Gamma j,\Gamma j+1}$;
    EN[k] $\leftarrow$ EN[k]+$E_{\Gamma j,\Gamma j+1}$;
    γ[k][i] $\leftarrow \Gamma_j$;
    i $\leftarrow$ i+1;
  end
end
`/* The tour finishes in the depot. */`
γ[k][i] $\leftarrow$ 0; SAct[k] $\leftarrow$ i;
`/* Put the first and the last subtour in 0 and 1. */`
(γ[0], EN[0], TM[0], SAct[0]) $\leftarrow$ (γ[1], EN[1], TM[1], SAct[1]);
(γ[1], EN[1], TM[1], SAct[1]) $\leftarrow$ (γ[k], EN[k], TM[k], SAct[k]);
`/* The index of the last sub-tour. */`
KAct $\leftarrow$ k-1;

4.2.1 Removal/Insertion operator

We develop a procedure to, from a decomposition γ of a tour Γ, remove and insert elements, obtaining a decomposition γ^* and therefore a tour Γ^*, trying to reduce the cost of $SP^{\alpha\beta}$. For doing this, we want to select the stations that are in a bad position and then analyze where insert them. To determine that a station is in a bad position we use the following two criteria:

- The first criteria is the increment of cost due the station j in the sub-tour. Let's be i_1, i_2 the stations (or depot/micro-plant) that are before and after j. The value of the first

criteria C_1^j is calculated as:

$$C_1^j = \frac{\alpha(T_{i1,j} + T_{j,i2} - T_{i1,i2}) + \beta(E_{i1,j} + E_{j,i2} - E_{i1,i2})}{\alpha T_{i1,i2} + \beta E_{i1,i2}}$$

This value is calculated for all stations except the ones which are the only station in the sub-tour, because in this case i_1, i_2 are the micro-plant or the depot and it can be a division by 0.

- The second criteria is only for the stations that are after or before the depot or micro-plant. In this case the value C_2 of this criteria is the cost between the station and the depot/micro-plant.

Let's say that n_{C_1} is the number of stations that we want to remove because of the first criteria and n_{C_2} for the second. The stations are sorted by their values of the criteria, and we select the n_{C_1} stations with the greatest value C_1 and the n_{C_2} with the greatest C_2. These stations are inserted in the sets R_1 and R_2 in the form of the input of the removal method. There can be repeated stations, but the removal method takes into account this fact and, if it isn't a station of R in the decomposition, R is updated without that station. So, at the end, we will get a decomposition without $\forall j \in R_1, R_2$ and $\forall i \in R_1, \forall j \in R_2$ $(i \neq j)$.

To get a new tour with all the stations,, we insert the eliminated stations in the best possible position in a given order. We give priority to the second criteria, so we start with the stations in R_2. For every possible position in the sub-tours, considering also the case of inserting the station in a new sub-tour, we calculate if it is feasible and its additional cost. It is feasible if $EN_\gamma \leq C^{Veh}$ in the case of the sub-tours that start and end in the micro-plant, where EN_γ is the needed energy for the sub-tour γ. In case of the sub-tour 0 and 1, we consider that it starts with E_0 energy units and it has to finish with at least E_0 units. The station is inserted in the feasible position with minimum additional cost. The Algorithm 6 shows the pseudocode of finding the best position to insert one station.

Using this process, for a decomposition γ it chooses always the same stations to remove and inserts them in a determined order. To give more flexibility to the process and generate different tours from the same decomposition, we add randomness to both processes.

For selecting the stations to remove, we will consider the $2 \cdot n_{C_1}$ and $2 \cdot n_{C_2}$ stations with the greatest C_1 and C_2, respectively. We insert each of the stations to the set R_1 or R_2 with a probability of the 50%. The values of n_{C_1} and n_{C_2} in our algorithm will depend on the number of stations of the instance. So, we add two parameters f_{C_1}, f_{C_2} that can take a value between 0 and 0.5, and we set $n_{C_1} = \lceil f_{C_1} * M \rceil$, $n_{C_2} = \lceil f_{C_2} * M \rceil$. The factor are limited to 0.5 to don't consider to remove more than the total number of stations.

For inserting the stations in a different order, the sets R_1 and R_2 are sorted randomly and the stations are introduced in the new order. We continue giving priority to the second criteria starting with the set R_2.

The procedure to find a good tour for the problem removing and inserting stations, called removal/insertion operator, appears in the Algorithm 7. The procedure receives as input the decomposition γ of the tour, and also the current tour $\Gamma Curr$, with its cost and refuelling transactions, that they are also the output of the procedure. When the procedure computes a

Algorithm 6: Find the position to insert the station j.

Input: γ, j
bestCost $\leftarrow \infty$; k $\leftarrow 0$;
while $k \leq KAct$ **do**
  i $\leftarrow 1$;
  while $i \leq SAct[k]$ **do**
    $\epsilon = E_{i-1,j} + E_{j,i} - E_{i-1,i}$;
    if $(k = 0 \wedge \epsilon + EN[k] \leq E_0) \vee (k = 1 \wedge \epsilon + EN[k] \leq C^{Veh} - E_0) \vee (k > 1 \wedge \epsilon + EN[k] < C^{Veh})$ **then**
      cost $\leftarrow \beta \cdot \epsilon + \alpha(T_{i-1,j} + T_{j,i} - T_{i-1,i})$;
      if *cost*<*bestCost* **then** bestCost$\leftarrow$cost; *bestK* $\leftarrow$k; *bestPos* $\leftarrow$i;
    end
  end
end
cost $\leftarrow \beta(E_{M+1,j} + E_{j,M+1}) + \alpha(T_{M+1,j} + T_{j,M+1} + p)$;
if *cost*<*bestCost* **then** Insert j in a new sub-tour;
else Insert j in the sub-tour *bestK*, in the position *bestPos*;

Algorithm 7: Remove and insert procedure to find a good tour.

Input: γ
Input/Output: $\Gamma Curr$, costCurr, refuelStCurr
$n_{C_1} = \lceil f_{C_1} \cdot M \rceil$; $n_{C_2} = \lceil f_{C_2} \cdot M \rceil$;
stop $\leftarrow$ False; i $\leftarrow 0$;
while $\neg stop \wedge i < s \cdot M$ **do**
  $R_1, R_2 \leftarrow$ selectStations$(\gamma, n_{C_1}, n_{C_2})$;
  $\gamma^* = \gamma$ removalStations(γ^*, R_1); removalStations(γ^*, R_2);
  performInsertion(γ^*, R_1, R_2);
  $\Gamma \leftarrow$ getTour(γ^*);
  refuelSt, cost $\leftarrow SP^{\alpha\beta}(\Gamma)$;
  if *cost* < *costCurr* **then**
    $\Gamma Curr \leftarrow \Gamma$; costCurr $\leftarrow$ cost;
    refuelStCurr $\leftarrow$ refuelSt; stop $\leftarrow$ True;
  end
  i $\leftarrow$ i+1;
end
return stop;

better tour, it returns True. The number of iterations that the procedure tries to get a better tour is bounded by a parameter s multiply by the number of stations of the instance.

Another procedure to try to get a better tour from a decomposition γ is to permute randomly the sub-tours (except the sub-tour 0 and 1), generate the tour Γ that corresponds to the new decomposition and then calculate the cost with $SP^{\alpha\beta}(\Gamma)$. This cost cannot be worst because the maximum cost is the previous solution as the sub-tours goes from the micro-plant to the micro-plant. The Figure 2 shows how a permutation of sub-tours can improve the the solution. The initial tour (a) is Depot $\rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow$ Depot, and the sub-tours of the decomposition are $\gamma_0 = [D, 3, MP]$, $\gamma_1 = [MP, D]$, $\gamma_2 = [MP, 1, MP]$, $\gamma_3 = [MP, 4, 2, MP]$, where D is the depot and MP is the micro-plant. After changing γ_2 and γ_3, we obtain the tour (b) $D \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow D$, and the $SP^{\alpha\beta}$ computes a solution with a better cost and different sub-tours: $\gamma_0 = [D, 3, 4, MP]$, $\gamma_1 = [MP, D]$,
$\gamma_2 = [MP, 2, 1, MP]$.

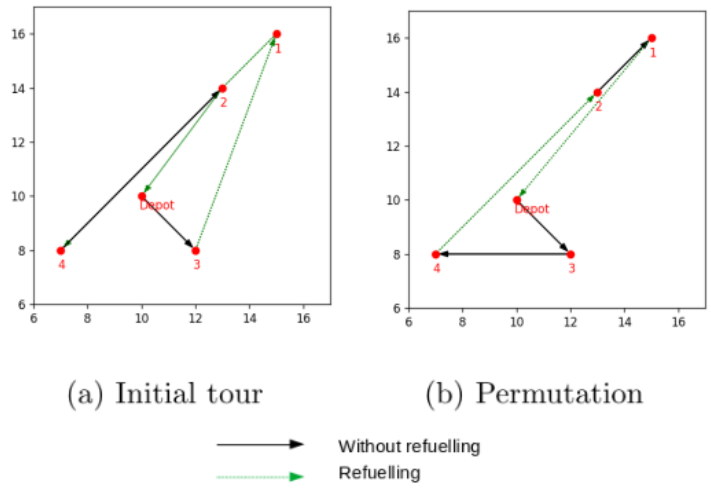


Figure 2: Example of a permutation of sub-tours that improve the result.

4.2.2 *Descent* algorithm

The global algorithm to find the tour for our problem can be see in the Algorithm 8, and we call it *Descent* algorithm. In the pseudocode, it uses first the procedure of removing and inserting stations and, if it doesn't find a better solution, it tries the procedure of permutation of sub-tours. If both procedures don't find a better solution, it stops.

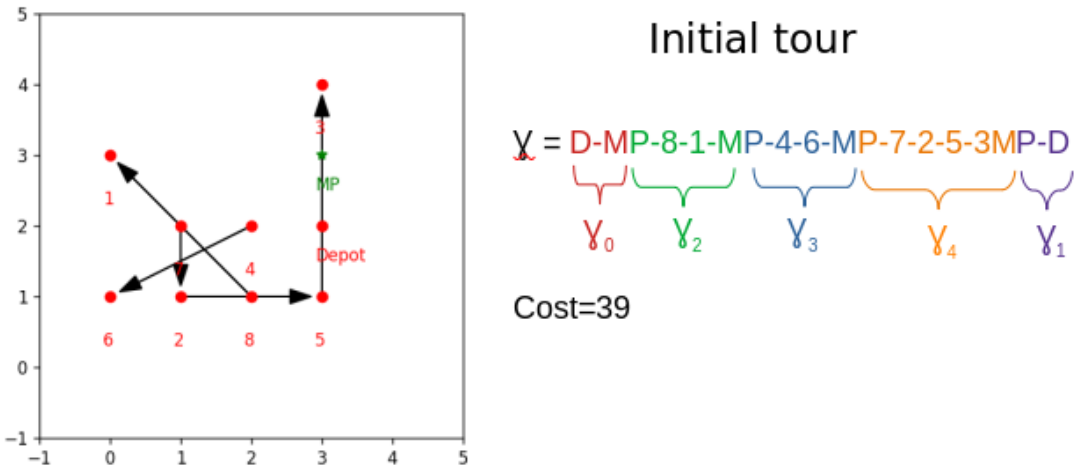
Instead of starting with a random tour, we will consider to use a good initial tour for the *Descent* algorithm. To construct the good tour, we will add randomly to the tour the first or the second nearest station to the previous one that hasn't be visited, starting from the depot. We will consider to run the *Descent* process with different initial tours constructed in this way and return the best final tour of all of them. This method is called Greedy Randomized Adaptive Search Procedure (GRASP), which allows to improve the performance of the heuristic in a easy way.

Algorithm 8: *Descent* algorithm to find a good tour.

```
/* Initialization Tour                                              */
ΓCurr is a tour with the stations randomly sorted;
refuelStCurr, costCurr ← SP^{αβ}(ΓCurr);
stop ← False;
while ¬stop do
    γ ← Decomposition(Γ,refuelStCurr);
    if ¬ procedureRemoveInsert(γ, ΓCurr, refuelStCurr, costCurr) then
        if ¬ procedurePermutation(γ, ΓCurr, refuelStCurr, costCurr) then
            stop=True;
        end
    end
end
```

4.2.3 Example

Now we are going to show an example of how the tour changes during the *Descent* algorithm. The Figure 3 shows this example, where D is the depot and MP the micro-plant. In the decomposition γ, the fact that MP elements are between two sub-tours means that the first finishes in the micro-plant and the next starts from there. It shows which stations were removed in each iteration and, in order, where they were inserted. It doesn't shows permutations because they didn't happen to find the solution in this example. The obtained solution isn't the optimal solution.



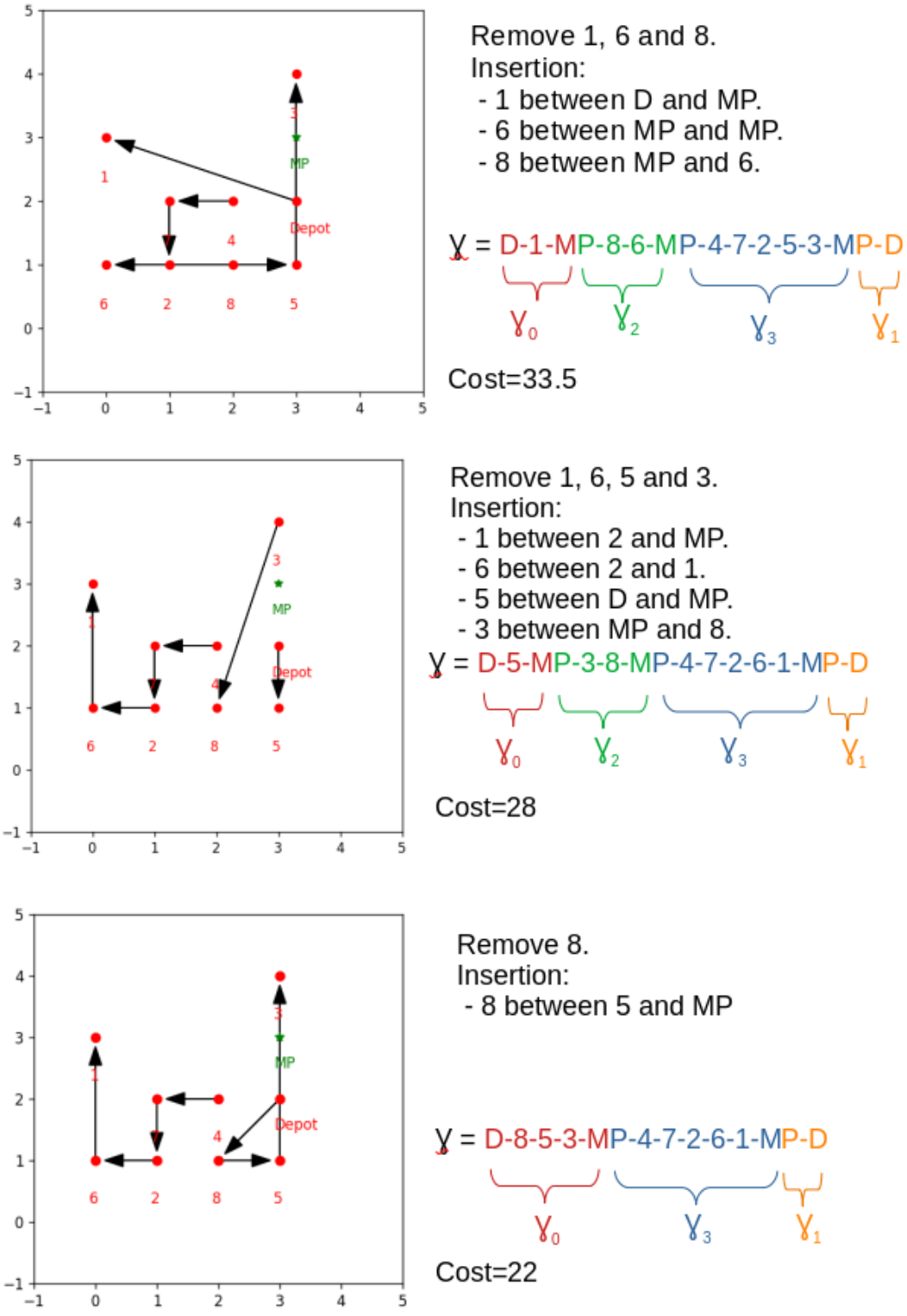


Figure 3: Evolution of the tour during the execution of the algorithm.

5 Generation of the instances

In this section, we explain how the instances of the problem are generated. The hydrogen is produced from period to period through a combination of photolysis and electrolysis, so the production is dependant of the weather. In order to generate instances close to the reality, we are going to divide the production in intervals, as the weather will be sunny, cloudy, etc in intervals, and also the cost of the electricity can be divided in intervals. These intervals can have highly efficient, efficient, medium, poor or very poor production rates; and they can be costly, medium or cheap periods.

For generating the instances, firstly we set the parameters of the instance generation process:

- N = Number of periods.
- M = Number of stations, without counting the *Depot* and the *micro-plat*.
- *Xlim*, *Ylim* = The limit of the coordinates of the stations in the X axis and Y axis respectively.
- K = The number of intervals, so the N periods will be divided in K intervals. For sake of simplicity, we are going to do in such a way that those intervals have the same number for periods. We add the parameter n, that will be the the number of periods of an interval. For this motive, N will not be a parameter and it will be calculated as N= n·K.
- Q expresses the stress induced by the energy requirements from the vehicle and the storage capacity of the vehicle to the number of refuelling transactions, so it is the expected number of transactions.
- H = The ratio between the hydrogen capacity of the production and the vehicle.
- R expresses the stress induced by the energy requirement from the vehicle to the production activity.
- S = The ratio between the part of the cost due to electricity consumption and the part of the cost due to the activation of the micro-plant.
- α = The conversion rate which converts time into cost value. This parameter will be calculate multiplying a coefficient C_α by the mean unit cost (see equation 1).

Secondly, we fix randomly the stations as points with integral coordinates in a 2-dimensional square of dimension *Xlim* $\times$ *Ylim*, where two stations cannot have the same coordinates. The micro-plant will be near the coordinates (x,y) of the *Depot*, exactly in the coordinates $(x,y+1)$. From that coordinates, the time matrix T is calculated as the value rounded up of the Euclidean distance between two stations, and the energy matrix as the value of the Manhattan distance. We calculate the expected energy E^* as the needed energy in a simple tour constructed in the next way. Starting from the *Depot* we select iterative the nearest not visited station until all stations are visited, returning at the end to the *Depot*. We also calculate the mean energy to go to the micro-plant E^{MP}_{mean} as $\frac{1}{M+1}\sum_{i=0}^{M} E_{i,-1}$.

Then, for every interval k, we randomly pick up 2 numbers Π_k in 1,...,5 and Θ_k in 1,...,3. For every period i of the interval k, we randomly generate an auxiliary production rate $RAux_i$ and a variable production costs $Cost^V_i$ according to the uniform distribution inside the intervals

$[\Pi_k/2, 3\Pi_k/2]$ and $[\Theta_k/2, 3\Theta_k/2]$ respectively. Next, we get the production rate $R_i = \lambda \cdot RAux_i$, by choosing λ in such a way that $\sum_i R_i =$R$\cdot E^*$. We calculate the mean unit cost:

$$C_{mean} = (\sum_i Cost_i^V / R_i)N \tag{1}$$

Finally, we set $C^{Veh} = E^*/Q + E^{MP}_{mean}$, we add E^{MP}_{mean} to take into account that the vehicle needs also capacity to go to refuel and for good tours the number of refuelling transactions is approximately Q. We set $C^{MP} = H \cdot C^{Veh}$ ans we set the fixed production cost as:

$$Cost^F = \frac{C_{mean} \cdot E^*}{2S \cdot (1 + E^*/C^{Veh})} \tag{2}$$

6 Machine learning approach

In this section, the second surrogate approach is explained. This one is based on using a neural network to approximate the cost of the SVREP problem for a giving tour Γ and its refuelling transaction.

For large instances, even the routing problem is too complex to be handle through ILP machinery. Therefore, we have to use the explained heuristic algorithm. From it, we obtain a collection γ of sub-tours $\gamma_0, \gamma_1, ..., \gamma_S$. This allow us to derive:

- The vectors Z and X of the global ILP program.
- The related stations of the refueling transactions, together with the vector L of the global ILP problem and the related loads $\mu_1, \mu_2, ..., \mu_S$. Notice that L and μ are the same, but with different indexation.
- For every refueling transaction s, the smallest period Min_s and the largest period Max_s, when it may take place.
- For any period $i = 1, ..., N$, we set $H_i^* = \sum_{\text{s such that i is possible for s}} \mu_s/(Max_s - Min_s + 1)$.

Fixing the vectors *Z, X* and L in the global ILP program, the production scheduling can be solved for that γ. Notice that, after fixing those variables, we can remove the constraints F1, F2, F2Bis, F3, F3Bis, F4, F4Bis, F5, F5, F7, F7Bis and F12. We will henceforth refer this reduce program as production ILP program. However, the *Descent* algorithm minimize the cost $\alpha \cdot Time + \beta \cdot Energy$, which isn't the cost of the EPC problem. Now, the idea is to train a neural network that approximates the cost value of the EPC problem, and use it during the *Descent* process. We change a bit the *Descent* algorithm in this case:

- We don't use the permutation operator, because we use the decomposition γ to get the cost and we don't need to retrieve the tour Γ until the end of the algorithm.
- In the remove/insert procedure (Algorithm 7), we obtain the cost of γ^* using the neural network instead of $SP^{\alpha\beta}$.

As matter of fact, we are not going to approximate the cost directly, but through a decomposition of the cost into a sum of:

- $Cost_{prod} = \sum_i z_i \cdot Cost_i^V$
- $Cost_{act} = \sum_i y_i \cdot Cost^F$
- $Cost_{time} = \alpha \cdot \tau_{M+1}$

Let us set:

- $Cmin = Inf_i \; Cost_i^V / R_i$
- $Cmax = Sup_i \; Cost_i^V / R_i$
- $Tmin = p \cdot Min_S + Time_S$
- $\mu = \sum_s \mu_s$

Then, the costs may be written as:

- $V_{prod} = \mu \cdot (\theta^{prod} \cdot Cmin + (1 - \theta^{prod}) \cdot Cmax)$
- $V_{act} = Cost^F(\theta^{act} + 10 \cdot (1 - \theta^{act}))$
- $V_{time} = \alpha \cdot (\theta^{time} \cdot Tmin + (1 - \theta^{time}) \cdot Tmax)$

We are going to learn the values of $\theta^{prod}, \theta^{act}, \theta^{time}$ through 3 neural networks. We suppose hear that $N = 20$ and we build those neural network according to this hypothesis. We discard the case when $Cost^F = 0$. Then we distribute the inputs in a vector of 62 entries, which represents 3 vector with indexation of the period set and 2 additional nodes:

- The first vector contains the coefficients $R_i/C^{MP}, i = 1, ..., 20$.
- The second vector the coefficients $Cost_i^V/Cost^F, i = 1, ..., 20$.
- The third vector has the values $H_i^*/C^{MP}, i = 1, ..., 20$.
- We add two nodes which receive inputs E_0/C^{MP} and $\alpha/Cost^F$.

Next we consider 4 intermediate layers, with respectively 20, 12, 8 and 4 nodes. The output layer contains 1 node. For this layers, we set the next synaptic arcs:

- **From the input layer to the first intermediate layer**.
  For every $i = 1, ..., 20$ we consider the set $K(i)$ of values k in 1,...,20 such that $i - 2 \leq k \leq i + 2$ and then the nodes $i, i + 20, i + 40$ of the input layer are connected to the nodes $k \in K(i)$ of the second layer. We also connect the nodes 61 and 62 of the input layer to every node of the second layer.
- **From the first intermediate layer to the second layer**.
  We connect every node $i = 1, ..., 20$ to any node k in 1,...,12 such that $|k - 3i/5| \leq 2$.

- **From the second intermediate layer to the third layer**.
  We connect every node $i = 1, ..., 12$ to any node k in 1,...,8 such that $|k - 2i/3| \leq 2$.

- **From the third intermediate layer to the fourth layer**.
  We connect every node $i = 1, ..., 8$ to any node k in 1,...,4.

- **From the fourth intermediate layer to the output layer**.
  We connect every node $i = 1, ..., 1$ to the output node.

The total number of synaptic arcs is $322 + 77 + 45 + 32 + 4 = 480$. The Figure 4 shows the layers of the neural network, except the input layer for better visibility as it has 62 nodes, but it is partial connected to the first layer as explained before.

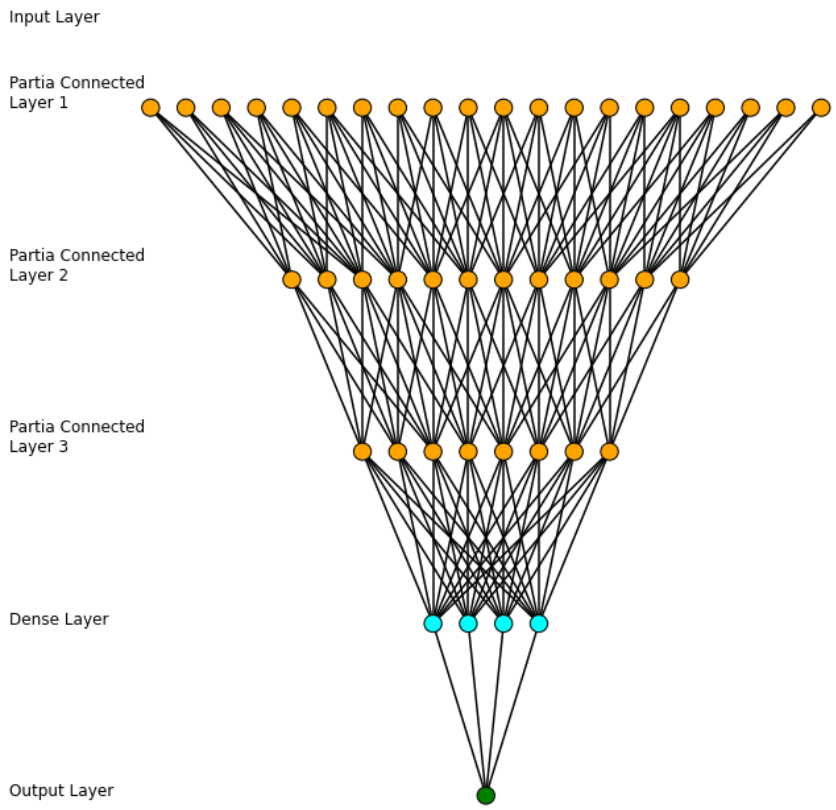


Figure 4: Diagram of the layers of the neural network

6.1 Neural network dataset

To generate the dataset, we should generate instances of the problem, and for each instance several tours with the refuelling transactions. Then, we obtain the time windows for the refuelling transactions and the vector H^*, to have the instance input for the neural network.

Next, we use the production ILP program to solve the global problem, and the values of θ are computed from its solution, that are the instance output.

We generate 10 groups of 50 instances, each group with the same parameters, which can be seen in the Table. For each instance, we generate one tour with the actual *Descent* algorithm. We want to generate several tours obtained by the initialization process of the *Descent* algorithm. However, there is a problem with those tours, because they have a high number of refills, and when it exceeds 10, the instances aren't feasible because they only have 20 periods and it isn't possible to produce enough energy (unless H and R are very high). For this motive, I decided to generate other 50 instances but reducing Q by 2 units, and for them generate generate 10 tours by that process and also other by the *Descent* algorithm. Therefore, the number of instances is $10 * (50 + 50 * 11) = 6000$. Even taking these precautions, 479 instances weren't feasible, so the dataset has 5521 instances.

However, these instances had a high time cost value, because as we want many feasible instances the value of p was set high in order to have a high maximum time. Therefore, we generate other 10 groups with the same parameters but with a lower p value (and the same consideration of Q for the simple tours). Adding the new feasible instances, we get a dataset of 10307 instances.

Group	**Stations**	**K**	**Q**	**H**	**R**	**S**	C_α	**p**
1	10	2	3	1.5	3	1	1	20
2	20	4	4	2	5	0.5	1.2	35
3	20	5	4	1	3	2	1.2	35
4	30	5	5	3	4	3	2	50
5	30	4	3	1	4	3	2	50
6	50	2	6	1.5	3.25	4	0.75	90
7	50	2	6	1.5	3.5	0.2	0.75	90
8	75	5	7	2.5	4.5	1	1.5	100
9	100	4	7	4	4	3.5	0.5	125
10	100	4	5	3	3	2	2	125
11	10	2	3	1.5	3	1	1	5
12	20	4	4	2	5	0.5	1.2	10
13	20	5	4	1	3	2	1.2	10
14	30	5	5	3	4	3	2	14
15	30	4	3	1	4	3	2	14
16	50	2	6	1.5	3.25	4	0.75	34
17	50	2	6	1.5	3.5	0.2	0.75	34
18	75	5	7	2.5	4.5	1	1.5	58
19	100	4	7	4	4	3.5	0.5	65
20	100	4	5	3	3	2	2	65

Table 2: Parameters of the instance groups.

Those instances are used for the training and validation processes. The validation process evaluates the performance of the neural network for instances not used during the training, but

they are used to set the hyperparameters of the neural network (number of epochs, batch size, optimizer, loss and activation functions). Therefore, we need a test dataset which will be used at the end of the training to check the final results of the neural network. We generate a new dataset of 1359 instances from the 20 groups.

6.2 Neural network results

The instances has been divided in two datasets, the train and the validation set. The train set contains the 80% of the instances, 9482, an the validation 825. Different models has been trained, with the hyperparameters that appear in the Table **??**. The number of epochs was chosen for each model depending when the validation loss stabilises. In the layers parameter, Partial connected means that the explained model was used, and Dense that the partial connected layers were substitute by dense layers (all nodes of one layer connected with all the nodes of the next one). We see in the results that it increases the training time without a benefit in the approximation.

After the model is trained, the three θ values are predicted for the instances of the validation test. Then, the time, production and activation costs are calculated and compared with the reals ones, obtaining the gaps of the Table 3.

Model	**Training time (s)** $\theta^T, \theta^P, \theta^A$	**Time gap**	**Production gap**	**Activation gap**	**Total gap**
1	33, 35, 83	5.52%	32.49%	18.47%	6.42%
2	22, 25, 87	10.93%	40.93%	48.6%	15.87%
3	22, 83, 83	4.72%	22.55%	16.7%	5.22%
4	64, 142, 106	4.46%	23.97%	16.3%	5.2%
5	123, 167, 208	4.05%	20.36%	15.49%	4.68%
6	125, 167, 263	4.44%	23.74%	17.2%	5.3%
7	231, 322, 443	4.25%	23.74%	16.24%	5.12%
8	223, 322, 76	3.87%	23.49%	64.09%	10.64%
9	262, 322, 160	4.35%	21.09%	17.27%	5.11%
10	323, 383, 183	4.41%	22.98%	17.17%	5.44%
11	503, 623, 304	4.13%	20.01%	16.62%	4.79%
12	512, 618, 307	7.82%	18.01%	16.36%	7.36%
13	562, 802, 442	4.28%	20.7%	16.05%	4.77%
14	503, 622, 299	3.86%	20.2%	17.93%	4.9%
15	522, 623, 305	4.04%	19.92%	16.28%	4.67%
16	442, 562, 322	4.83%	28.78%	19.02%	5.83%

Table 3: Results of the models.

Other fact that have been notice, is that the gap between the predicted values of θ^P and the real ones are less different than the gap in θ^T, although in the costs is the contrary. That means that small changes in θ^P produces bigger changes in the cost that the θ^T, and a similar effect happens in θ^A.

We decide that our final model will be the one that gives the best result for each neural network. That means that the time model will be the 14, the production model the 12 and the activation model the 5. To get the final gaps we use the test dataset, and we obtain a time gap of 11.2%, a production gap of 21.57%, an activation gap of 18.95% and a total gap of 4.32%. Although the gaps have increased, the total cost gap is bellow the 5%, which is a good approximation for a complex value that depends of many factors.

Model	**Activation function**	**Layers**	**Batch size**	**Epoch** $\theta^T, \theta^P, \theta^A$	**Loss**	**Optimizer**
1	sigmoid(1)	Partial connected	16	25, 25, 50	mean squared error	Adam
2	sigmoid(0.5)	Partial connected	16	10, 10, 50	mean squared error	Adam
3	sigmoid(2)	Partial connected	16	20, 50, 50	mean squared error	Adam
4	sigmoid(2)	Partial connected	8	30, 40, 50	mean squared error	Adam
5	sigmoid(2)	Partial connected	4	30, 40, 50	mean squared error	Adam
6	sigmoid(1)	Partial connected	4	30, 40, 50	mean squared error	Adam
7	sigmoid(1)	Partial connected	2	30, 40, 50	mean squared error	Adam
8	relu output sigmoid(1)	Partial connected	2	30, 40, 10	mean squared error	Adam
9	sigmoid(2)	Partial connected	2	30, 40, 20	mean squared error	Adam
10	sigmoid(2)	Dense	2	30, 40, 20	mean squared error	Adam
11	sigmoid(2)	Partial connected	1	30, 40, 20	mean squared error	Adam
12	sigmoid(2)	Partial connected	1	30, 40, 20	mean absolute error	Adam
13	sigmoid(2)	Partial connected	1	30, 40, 20	mean squared logarithmic error	Adam
14	relu output sigmoid(2)	Partial connected	1	30, 40, 10	mean squared error	Adam
15	sigmoid(2)	Partial connected	1	30, 40, 20	mean squared error	NAdam
16	sigmoid(2)	Partial connected	1	30, 40, 20	mean squared error	Adamax

Table 4: Hyperparameters of the models.

7 Experimentation

In this section, we present the results of the proposed methods to solve the SVREP problem. First, we study the behaviour of the global ILP model which solves the complete problem. The parameters of the 10 instances used for this experimentation are shown in the Table 5. These parameters are explained in the section 5.

Instance	**Periods**	**Stations**	**K**	**Q**	**H**	**R**	**S**	C_α	**p**	C_{mean}
1	20	6	5	2	1	2	1	1.5	4	9.74
2	20	10	4	2	1	3	3	0.5	5	6.07
3	20	10	2	3	1.5	3	1	1	5	4.88
4	20	15	4	3	2	2.5	2	1	5	4.42
5	20	20	4	4	2	5	0.5	1.2	10	1.57
6	20	20	5	4	1	3	2	1.2	10	2.32
7	20	30	5	5	3	4	3	2	14	0.88
8	20	30	4	3	1	4	3	2	14	1.81
9	20	50	2	6	1.5	3.25	4	0.75	34	0.39
10	20	50	2	6	1.5	3.5	0.2	0.75	34	0.66

Table 5: Parameters of the instances.

The results of the global ILP model for these instances are computed, while not allowing more than 1 hour of execution. The Table 6 shows the retrieved values: the lower bound, the upper bound, the CPU time (if it finds the optimal solution), the root relaxation values and the number of *Extended No Subtour* cuts which are generated. In some instances it didn't even find a feasible integer solution. We see that it can only solve small instances, and the gap between the bounds is considerably high.

Instance	**Lower bound**	**Upper bound**	**CPU (s)**	**Relaxation**	**Cuts**
1	1555.6	1555.6	122.09	582.45	64
2	117.709	269.92	-	93.097	1037
3	248.88	674.68	-	198.836	343
4	214.466	416.4	-	192.718	2115
5	160.799	638.76	-	149.488	2825
6	234.433	627.24	-	217.322	3169
7	-	-	-	-	-
8	335.759	937.52	-	327.901	2215
9	-	-	-	-	-
10	-	-	-	-	-

Table 6: Results of the global ILP model .

For those instances, we solve the vehicle routing problem with the ILP surrogate model and the heuristic algorithm. For both methods, we use three values of β (the conversion rate of energy into cost). The value β_0 is the mean unit cost of equation 1.

The results computed by the ILP surrogate model appears Table 7. As with the global model, we set the limit of computational time to 1 hour and the separation algorithm callback is only executed one in 50 times. The second part of the table shows the separation of the costs an the number of refills.

Instance	β	**Lower bound**	**Upper bound**	**CPU (s)**	**Relaxation**	**Cuts**
1	β_0	1324.09	1324.09	0.04	1118.34	11
	$\beta_0/4$	944.222	944.222	0.04	792.275	6
	$4\beta_0$	2843.56	2843.56	0.04	2406.82	9
2	β_0	352.332	352.332	0.09	320.929	9
	$\beta_0/4$	188.403	188.403	0.12	167.824	10
	$4\beta_0$	1008.05	1008.05	0.14	927.704	10
3	β_0	600.07	600.081	0.37	501.397	30
	$\beta_0/4$	395.24	395.24	0.33	327.062	31
	$4\beta_0$	1419.44	1419.44	0.26	1176.02	23
4	β_0	486.269	486.269	0.09	470.417	6
	$\beta_0/4$	320.467	320.467	0.06	314.574	6
	$4\beta_0$	1149.48	1149.48	0.07	1105.3	7
5	β_0	409.191	453.564	-	352.684	1933
	$\beta_0/4$	292.187	322.071	-	248.826	2059
	$4\beta_0$	902.824	979.537	-	765.318	1949
6	β_0	542.076	542.122	17.42	479.892	296
	$\beta_0/4$	378.255	378.26	5.79	336.95	118
	$4\beta_0$	1197.48	1197.57	7.02	1054.57	135
7	β_0	390.529	438.129	-	376.73	2560
	$\beta_0/4$	313.071	352.412	-	289.539	2590
	$4\beta_0$	695.417	775.721	-	660.389	1650
8	β_0	633.548	700.362	-	604.197	2411
	$\beta_0/4$	491.733	554.141	-	473.378	1795
	$4\beta_0$	1182.06	1274.42	-	1110.02	1947
9	β_0	153.872	189.826	-	149.599	2071
	$\beta_0/4$	107.437	131.507	-	105.305	2555
	$4\beta_0$	357.85	422.618	-	345.149	2153
10	β_0	237.736	285.417	-	231.836	2107
	$\beta_0/4$	159.862	196.805	-	155.75	2322
	$4\beta_0$	550.771	644.004	-	538.169	2419

Table 7: Results of the surrogate ILP program.

We solve the instances with the heuristic algorithm, using 10 GRASP initialization and 2M trials per iteration of the descent process, where M is the number of stations. The Table 9 shows the cost value, the CPU time, the mean number of iterations per initialization and the mean number of trials per iteration. We see that for the length of these instances, the *Descent* algorithm finds the optimal solution very fast and with few iterations. In the table 8, we show the same values, in this case the cost value is the one approximate by the neural network for

the final tour. In this case it takes more CPU time because the program needs to load the Python libraries to use the neural network.

Instance	**Cost**	**CPU (s)**	**Iterations**	**Trials**
1	1723.1	1.41314	1.6	8.81
2	615.785	1.43209	2.9	8.48
3	856.584	1.43222	3	8.3
4	624.123	1.45348	4.2	8.57
5	358.051	1.50965	4.4	12.1
6	499.106	1.48206	5.1	9.1
7	448.105	1.62965	6.1	12.2
8	836.66	1.5856	6.1	13.4
9	259.071	1.79791	1	100
10	407.478	1.86288	7.8	15.9

Table 8: Results of the *Descent* algorithm using the neural network.

We want the production schedule for the tours obtained from these methods, and compare the solutions with the ones of the global ILP model. For doing this, we deduce from the tour Γ and the refuelling transitions the decomposition of sub-tours γ. From that decomposition we can deduce the matrices Z and X of the ILP model. Moreover, if we consider that the vehicle only refuels the enough quantity of hydrogen for the next tour (taking into account that it has to end with at least E_0), we can deduce the vector L, the amount of refuelled load. We set these values in the global ILP model, that becomes a linear program of only the production scheduling.

The Table 10 shows the results obtained from the tours computed by the three methods. We denote by *W-ILP* the global cost computed from the solution of the surrogate ILP program, by *W-Heur* the one from the solution of the *Descent* algorithm approximating the cost with the $SP^{\alpha\beta}$ algorithms and by *W-ML* the one from the solution that uses the neural network. The last one doesn't depend on the parameter β. We also see that solving the tour routing with the *Descent* algorithm and then the production problem with linear programming allows us to solve the problem very fast obtaining a solution near the optimal.

The Table 11 shows the characteristics of the solutions of the global problem, obtained from the tours of the surrogate ILP program using $\beta = \beta_0/4$. The Table shows the number of refills and activations of the micro-plant, the time **T-Veh** when the vehicle reaches the end of the tour, the time **T-MP** when the micro-plant ends its production process, the division of the cost and the mean energy unit cost for the solution. We see that in general the consumed time by the vehicle is much greater than the consumed energy, so it means that it have to wait many time for the refuelling transactions. Therefore, it is normal that the solution C_{mean} is greater than the C_{mean} of the instance because it has to produce in some costly periods in order for the vehicle to finish the tour on time.

Instance	β	**Cost**	**CPU (s)**	**Iterations**	**Trials**
1	β_0	1324.09	0.008544	3.1	7.74
	$\beta_0/4$	944.222	0.007683	3.1	6.87
	$4\beta_0$	2843.56	0.008249	3.6	6.44
2	β_0	352.332	0.01346	4.1	6.95
	$\beta_0/4$	188.403	0.012418	5.3	5.77
	$4\beta_0$	1008.05	0.01046	4.3	6.3
3	β_0	600.081	0.015922	3.9	10.1
	$\beta_0/4$	395.24	0.016386	4.6	9.13
	$4\beta_0$	1419.44	0.018737	5.1	9.96
4	β_0	486.269	0.030216	5	10.8
	$\beta_0/4$	320.467	0.025735	4.5	9.91
	$4\beta_0$	1149.48	0.023203	4.2	9.81
5	β_0	453.564	0.080386	7.1	14.7
	$\beta_0/4$	322.071	0.066201	6.1	13.6
	$4\beta_0$	979.537	0.08353	8.1	13
6	β_0	595.128	0.075975	7.8	11.9
	$\beta_0/4$	420.807	0.072303	8	11.1
	$4\beta_0$	1292.41	0.058402	6.3	11.2
7	β_0	439.887	0.303306	11.3	18.5
	$\beta_0/4$	352.852	0.295766	10.2	19.1
	$4\beta_0$	777.481	0.328499	12.4	17.1
8	β_0	714.802	0.181128	8.2	14.9
	$\beta_0/4$	565.873	0.133639	7.3	12
	$4\beta_0$	1299.69	0.140364	6.2	15.4
9	β_0	192.056	1.01516	17.6	21.1
	$\beta_0/4$	133.253	1.34089	20	22.8
	$4\beta_0$	437.637	1.28821	19.3	25.4
10	β_0	295.687	0.658925	14.1	18.4
	$\beta_0/4$	196.805	0.657488	13.2	20.1
	$4\beta_0$	639.201	0.492755	14.2	13.6

Table 9: Results of the *Descent* algorithm.

Instance	β	**W-ILP**	**W-Heur**	**W-ML**
1	β_0	1700.6	1594.6	1594.6
	$\beta_0/4$	1594.6	1594.6	
	$4\beta_0$	1700.6	1594.6	
2	β_0	331.92	331.92	331.92
	$\beta_0/4$	331.92	331.92	
	$4\beta_0$	331.92	331.92	
3	β_0	669.68	687.28	701.68
	$\beta_0/4$	669.68	704.68	
	$4\beta_0$	669.68	669.68	
4	β_0	416.4	416.4	416.4
	$\beta_0/4$	416.4	416.4	
	$4\beta_0$	416.4	416.4	
5	β_0	612.76	638.76	612.76
	$\beta_0/4$	632.56	638.76	
	$4\beta_0$	612.76	612.76	
6	β_0	514.43	588.34	627.24
	$\beta_0/4$	514.43	627.24	
	$4\beta_0$	514.43	588.34	
7	β_0	486.08	487.84	485.96
	$\beta_0/4$	486.08	487.84	
	$4\beta_0$	450.24	479.72	
8	β_0	726.97	733.36	733.36
	$\beta_0/4$	726.97	737.8	
	$4\beta_0$	742.97	726.97	
9	β_0	222.738	220.659	-
	$\beta_0/4$	221.844	214.659	
	$4\beta_0$	222.738	229.341	
10	β_0	757.446	764.949	757.446
	$\beta_0/4$	767.446	766.949	
	$4\beta_0$	759.949	767.446	

Table 10: Results of the production ILP program fixing the tour.

Instance	**Refills**	**Activations**	**T-Veh**	**T-MP**	$Cost_{act}$	$Cost_{prod}$	$Cost_{time}$	C_{mean}
1	3	4	71	80	268	290	1036.60	15.26
2	2	3	48	85	48	138	145.92	6.90
3	4	4	86	95	148	102	419.68	7.85
4	3	3	70	75	69	38	309.40	2.11
5	5	4	187	190	200	81	351.56	3.00
6	3	5	117	190	70	118	326.43	3.81
7	5	4	208	210	32	88	366.08	3.67
8	3	3	177	196	33	55	638.97	1.28
9	6	6	484	510	18	63	140.84	1.75
10	5	5	518	578	410	100	257.45	2.50

Table 11: Characteristics of the solutions from the tour of the surrogate ILP program with $\beta = \beta_0/4$.