Dynamic programming for the single unit hydro unit commitment problem

Alexandre Heintzmann, Christian Artigues, Pascale Bendotti, Sandra Ulrich Ngueveu, Cécile Rottner

Dynamic programming for the single unit hydro unit commitment problem

Alexandre Heintzmann[1,2] Christian Artigues[2] Pascale Bendotti[1]
Sandra Ulrich Ngueveu[2] Cécile Rottner[1]

[1]EDF Lab Paris-Saclay, 7 Bd. Gaspard Monge, 91120 Palaiseau, France
{alexandre.heintzmann,pascale.bendotti,cecile.rottner@edf.fr}
[2]LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
{alexandre.heintzmann,christian.artigues,sandra.ulrich.ngueveu@laas.fr}

Abstract

The Hydro Unit Commitment problem (HUC) specific to hydroelectric units is part of the electricity production planning problem, called Unit Commitment Problem (UCP). More specifically, the case studied is that of the HUC with a single unit, denoted 1-HUC. The unit is located between two reservoirs. The time is discretized in time periods. The unit operates at a finite number of points defined as pairs of the generated power and the corresponding water flow. Several constraints are considered. Each reservoir has an initial volume, as well as a maximum and minimum volume per time period. The unit has both ramping and min-up/min-down constraints limiting the flow rate variation between two consecutive time periods. At each time period, there is an additional positive, negative or zero intake of water in the reservoirs. The case of a price-taker revenue maximization problem is considered. An efficient A* algorithm is first proposed to solve a simplified variant of the 1-HUC without ramping nor min-up/min-down constraints. The algorithm is very efficient to solve difficult instances and is shown to have a more stable behavior than CPLEX on realistic EDF instances. For the complete variant of the 1-HUC including both the ramping and min-up/min-down constraints the A* algorithm, as well as standard resource-constrained shortest path approaches are shown to be ineffective. The latter change is due to the weakening of the dominance rules. To overcome this difficulty, we propose a two-phase algorithm inspired by algorithms used in bi-criteria optimization. The first phase solves extremely fast simple shortest path problems while the second phase is based on a K-best solutions algorithm to enumerate feasible solutions in a restricted search space. We compare the dedicated two-phase algorithm with a mixed-integer linear program solved by CPLEX and show that the proposed algorithm outperforms CPLEX on instances with 48 time periods, 20 operating points and a small price variability.

1 Introduction

The single unit Hydro Unit Commitment (1-HUC) is defined as follows. Let a valley consist of a unit and two reservoirs, the unit being located between the two reservoirs. The principle of hydroelectric

production is as follows: the water from the upstream reservoir flows into the downstream reservoir through the unit, thus driving the turbines of the unit which in turn powers the generator to produce electricity. The unit operates at a finite number of points in $I = \{1, \ldots, M\}$. Each operating point $i \in I$ is defined as a pair formed by a water flow D_i and a generated power P_i. The operating points are defined in a cumulative fashion, i.e., if a unit is at operating point i, then order constraints apply involving all points $j < i$ to also be operated. The time horizon is discretized into T time periods. At each time period t, the unit turbines a water flow and produces an amount of energy that is considered to be constant for the duration of the time period. Each reservoir n has a maximum capacity $\overline{V}_t^n$ and a minimum capacity $\underline{V}_t^n$ that are time-dependent. Water management policies require that reservoirs should meet target volumes at some specified time periods. The latter requirement is captured in the time-dependent minimum/maximum capacity. At each time period, the reservoir n receives an additional intake of water A_t^n, being positive, negative or null.

The revenues take into account the unit value Φ^n of the water in each reservoir n at the end of the time horizon, and the resale of the energy produced at a unit value Λ_t variable over time. The problem consists in maximizing the total revenue, by turbining while satisfying the reservoir capacities at each time period.

With $x_{t,i}$ the binary variable indicating whether the unit is at least at operating point i at time period t, we obtain the following formulation:

$$\max \sum_{t=1}^{T}\sum_{i=1}^{M} \Lambda_t P_i x_{t,i} + \Phi^1\Big(\sum_{t=1}^{T}(A_t^1 - \sum_{i=1}^{M} D_i x_{t,i})\Big) + \Phi^2\Big(\sum_{t=1}^{T}(A_t^2 + \sum_{i=1}^{M} D_i x_{t,i})\Big)$$

$$\text{s.c.} \quad V_0^1 + \sum_{t=1}^{k}(A_k^1 - \sum_{i=1}^{M} D_i x_{t,i}) \leq \overline{V}_k^1, \qquad \forall k \leq T \qquad \text{(a1)}$$

$$V_0^1 + \sum_{t=1}^{k}(A_k^1 - \sum_{i=1}^{M} D_i x_{t,i}) \geq \underline{V}_k^1, \qquad \forall k \leq T \qquad \text{(b1)}$$

$$V_0^2 + \sum_{t=1}^{k}(A_k^2 + \sum_{i=1}^{M} D_i x_{t,i}) \leq \overline{V}_k^2, \qquad \forall k \leq T \qquad \text{(a2)}$$

$$V_0^2 + \sum_{t=1}^{k}(A_k^2 + \sum_{i=1}^{M} D_i x_{t,i}) \geq \underline{V}_k^2, \qquad \forall k \leq T \qquad \text{(b2)}$$

$$x_{t,i} \geq x_{t,i+1}, \qquad \forall t \leq T, \forall i \leq M-1 \qquad \text{(c)}$$

$$x_{t,i} \in \{0,1\}, \qquad \forall t \leq T, \forall i \leq M \qquad \text{(d)}$$

1.1 Rewriting the inequalities

In order to better handle the inequalities, we will rewrite the inequalities $(a1)$, $(a2)$, $(b1)$ and $(b2)$ as follows:

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \geq V_0^1 + \sum_{t=1}^{k} A_t^1 - \overline{V}_k^1, \qquad \forall k \leq T \tag{a1}$$

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \leq V_0^1 + \sum_{t=1}^{k} A_t^1 - \underline{V}_k^1, \qquad \forall k \leq T \tag{b1}$$

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \leq \overline{V}_k^2 - V_0^2 - \sum_{t=1}^{k} A_t^2, \qquad \forall k \leq T \tag{a2}$$

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \geq \underline{V}_k^2 - V_0^2 - \sum_{t=1}^{k} A_t^2, \qquad \forall k \leq T \tag{b2}$$

There are redundancies between inequalities $(a1)$ and $(b2)$, and between inequalities $(a2)$ and $(b1)$. Let us introduce β_k and α_k in the following way:

$$\beta_k = \max(V_0^1 + \sum_{t=1}^{k} A_t^1 - \overline{V}_k^1 \quad , \quad \underline{V}_k^2 - V_0^2 - \sum_{t=1}^{k} A_t^2) \qquad \forall k \leq T$$

$$\alpha_k = \min(V_0^1 + \sum_{t=1}^{k} A_t^1 - \underline{V}_k^1 \quad , \quad \overline{V}_k^2 - V_0^2 - \sum_{t=1}^{k} A_t^2) \qquad \forall k \leq T$$

We can replace the inequalities $(a1)$, $(a2)$, $(b1)$ and $(b2)$ with the following inequalities:

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \geq \beta_k \qquad \forall k \leq T \tag{a}$$

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \leq \alpha_k \qquad \forall k \leq T \tag{b}$$

Inequalities (a) are nested knapsack constraints, and we define (b) as a nested inverted knapsack constraints.

1.2 Rewriting the objective function

Noting that the constraints of our problem are knapsack constraints with order constraints, we rewrite the objective function to look like an objective function of a knapsack problem.

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{i=1}^{M} \Lambda_t P_i x_{t,i} + \Phi^1\Big(\sum_{t=1}^{T}(A_t^1 - \sum_{i=1}^{M} D_i x_{t,i})\Big) + \Phi^2\Big(\sum_{t=1}^{T}(A_t^2 + \sum_{i=1}^{M} D_i x_{t,i})\Big) \\
=&\sum_{t=1}^{T}\sum_{i=1}^{M} \Lambda_t P_i x_{t,i} + \Phi^1 \sum_{t=1}^{T} A_t^1 - \sum_{t=1}^{T}\sum_{i=1}^{M} \Phi^1 D_i x_{t,i} + \Phi^2 \sum_{t=1}^{T} A_t^2 + \sum_{t=1}^{T}\sum_{i=1}^{M} \Phi^2 D_i x_{t,i} \\
=&\sum_{t=1}^{T}\sum_{i=1}^{M} (\Lambda_t P_i - \Phi^1 D_i + \Phi^2 D_i) x_{t,i} + \Phi^1 \sum_{t=1}^{T} A_t^1 + \Phi^2 \sum_{t=1}^{T} A_t^2
\end{aligned}$$

The objective function is therefore to maximize the value of each active operating point, plus a constant.

Using (a), (b) and the rewritten objective function, we obtain a compact formulation:

$$\max \sum_{t=1}^{T}\sum_{i=1}^{M}(\Lambda_t P_i - \Phi^1 D_i + \Phi^2 D_i)x_{t,i} + \Phi^1 \sum_{t=1}^{T} A_t^1 + \Phi^2 \sum_{t=1}^{T} A_t^2$$

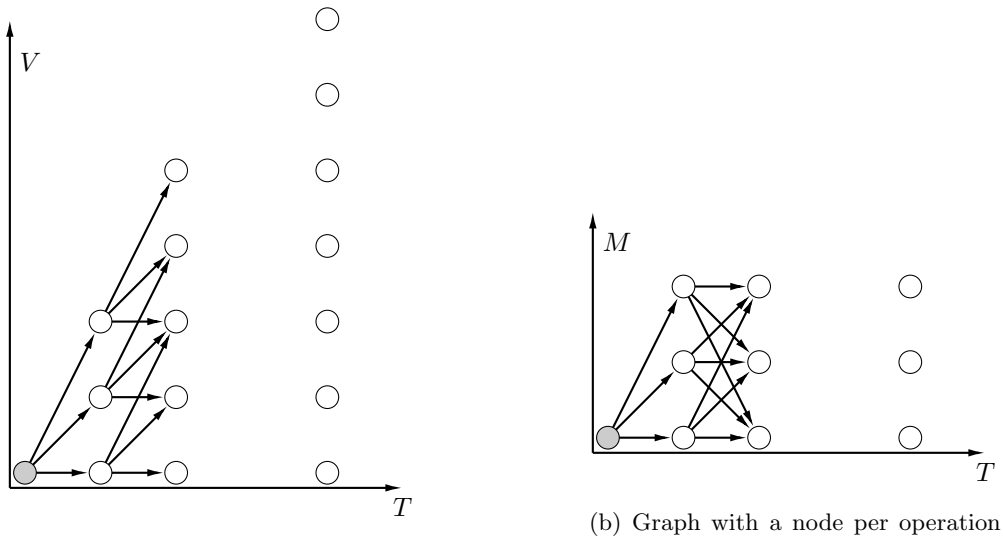
$$\text{s.c.} \quad \sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \geq \beta_k \qquad \forall k \leq T \qquad \text{(a)}$$

$$\sum_{t=1}^{k}\sum_{i=1}^{M} D_i x_{t,i} \leq \alpha_k \qquad \forall k \leq T \qquad \text{(b)}$$

$$x_{t,i} \geq x_{t,i+1} \qquad \forall t \leq T, \forall i \leq M-1 \qquad \text{(c)}$$

$$x_{t,i} \in \{0,1\} \qquad \forall t \leq T, \forall i \leq M \qquad \text{(d)}$$

The 1-HUC can be represented graphically in two ways. **Figure 1**(a) illustrates the case where each node represents, for a partial solution, the amount of volume debited up to a given time period. **Figure 1**(b) illustrates the case where each node represents, for a partial solution, the highest operating point used at a given time period. In both cases the shaded node is the source node. The first case is similar to the representation of a knapsack problem, while the second case looks like the representation of a shortest (or longest) path problem. In both cases, the underlying algorithm for solving the problem via dynamic programming is the shortest path algorithm. For the second representation, it is necessary to keep the volume in memory so as not to generate invalid solutions.



(a) Graph with a node per volume and per time period

(b) Graph with a node per operation point and per time period

Figure 1: Graphical representations of the 1-HUC

1.3 Additional constraints

In practice, the 1-HUC has additional constraints, among which the ramping and min-up/down constraints.

The min-up/min-down constraints are defined as follows [12]. Each unit stepping up (down, resp.) point i must operate at point $j \geq i$ ($j < i$, resp.) for at least L_i (l_i, resp.) time periods after stepping up (down resp.). The associated constraints are as follows.

$$x_{t,i} - x_{t-1,i} \leq x_{t',i} \qquad \forall t' \in [t+1, \min(t+L_i, T)], \forall i, \forall t \in [2, T] \tag{e}$$

$$x_{t-1,i} - x_{t,i} \leq 1 - x_{t',i} \qquad \forall t' \in [t+1, \min(t+l_i, T)], \forall i, \forall t \in [2, T] \tag{f}$$

The ramping constraints indicate that the upward flow variation is at most RU per time period, and symmetrically the downward flow variation is at most RD per time period. If a volume of water v is discharged at a time period t, the volume of water discharged at time period $t+1$ must lie in the interval $[v - RD; v + RU]$. These constraints can be expressed by the following inequalities.

$$\sum_{i=1}^{M} D_i x_{t,i} - \sum_{i=1}^{M} D_i x_{t-1,i} \leq RU \qquad \forall t \in [2; T] \tag{g}$$

$$\sum_{i=1}^{M} D_i x_{t-1,i} - \sum_{i=1}^{M} D_i x_{t,i} \leq RD \qquad \forall t \in [2; T] \tag{h}$$

In this paper, we consider the 1-HUC defined by the constraints (a) to (h).

2 Literature review

2.1 Dynamic programming for the Hydro Unit Commitment

In this first part, we focus on dynamic programming algorithms for the HUC, which are partly cited in the survey of Raouia Taktak and Claudia D'Ambrosio [16].

In [2] a formulation to solve the HUC on instances of the Itaipù unit (Brazil, Paraguay) with dynamic programming is described. This formulation minimizes a cost function that takes into account the cost of starting and stopping the turbines, as well as the cost of the energy losses of the turbines. The only constraint is to satisfy the minimum and maximum number of turbines running at each time period. This formulation differs from ours because there is no volume considered, therefore no target volume, and there are limits on the number of turbines running at each time period, which is a constraint we do not consider. Also, it is specified that the 18 turbines of Itaipù unit are identical, while the operating points that we use are not necessarily identical. The dynamic programming algorithm is not described and no heuristics are described.

In [10] a method for solving a nonlinear 1-HUC is described. The first non-linearity considered is the effect of head on the efficiency of the unit, and on the maximum and minimum flow rate. The second non-linearity considered is the power as a function of head and flow rate. For this 1-HUC, there is a target volume to achieve. To solve this problem with dynamic programming, a state diagram is constructed as follows. The volume is discretized into equidistant reachable volumes. The target volume is relaxed to be reachable with the volume discretization. The state diagram is constructed by

generating the possibilities to reach the target volume from the initial volume, respecting the upper and lower bounds on the volume at each time period. Starting from the state at the end of the time horizon, the dynamic programming algorithm maximizes the value of the generated power. Unlike the 1-HUC we consider, the 1-HUC tackled in [10] does not contain operating points, but a discretization of the volume. The number of volume possibilities is proportional to this discretization, while it becomes exponential when using operating points. It is possible to have a much finer discretization than the number of operating points, and one could consider that the two 1-HUC are equivalent if the discretization is fine enough with respect to the flow rates of the operating points. However, in the given results, it is specified that the discretization is 0.3% to 0.5% of the difference between the minimum and maximum volume of the reservoir. That is, between 200 to 300 possibilities per time step, whereas for the 1-HUC with operating points, only 5 operating points can lead to many more possibilities if the reservoirs are large. Another difference is that the target volume can be relaxed, which is not the case in our study. More precisely, it is assumed that 1-HUC instances are feasible. If it were not the case due to conflicting constraints between target volume and operating on discrete points, then target volume could be adjusted [15]. The last difference is that instances of our 1-HUC do not necessarily have a target volume, so there may be a large number of possible volumes in the time horizon. This would be the case if no water management requirements apply within the time horizon. An algorithm starting from the last time period would not be applicable to our case, unless it is used for any reachable volume, which is hardly possible. This dynamic programming algorithm is used in [11] to find a scatter plot which is then used in a piecewise linear formulation.

In [1] a decomposition method for solving the HUC with shortest paths is described. The HUC considered is a valley where each unit has a finite number of operating points. The topology of the valley is not restricted to a chain, as each unit (resp. reservoir) can have a set of upstream and downstream reservoirs (resp. units). For this HUC, the units can also involve pumps. To represent pumping, the operating points can have negative flow and power. The operating points are considered in a disjunctive fashion, namely the constraints indicate that a unit can only be at one operating point per time period. Using negative values for operating points also ensures that a unit cannot pump and turbine simultaneously. The constraints of this HUC take into account additional ramping constraints: the flow variation is limited from one time period to another. This HUC also takes into account a target volume for each reservoir, at the last time period. Note that the latter target volume is a minimal bound, meaning there is no equality constraint. The criterion to be maximized is the valley efficiency, taking into account: the price of the generated and purchased power (for pumping) at each time period and the startup price of the turbines. The solution approach decomposes this HUC into multiple 1-HUC. Each 1-HUC is solved either by a shortest path algorithm, or by a label algorithm adapted from the one used to solve the Resource-Constrained Shortest Path Problem (RCSPP). Therefore, a shortest path graph representation is used (see Figure 1(b)). The difference between the RCSPP and the 1-HUC lies in the resource constraints : there are both an upper and a lower bound on the amount of volume in the reservoir in the 1-HUC, whereas only an upper bound in the RCSPP. A first difficulty is that the volume can increase and decrease over time, especially if the instance considered is a pumped storage power station that can transfer energy by pumping water from the downstream reservoir to the upstream reservoir. A fictitious volume is defined, by increasing the flow rate of the operating points by the maximum flow rate that can be pumped in a single time period. A similar

shift is done for the operating points, which now have non-negative water flows. In order to keep the problem the same, the bounds on the volume have also to be increased. Note that this modification is not necessary in the case of a unit that can only involve turbines. The evolution of the fictitious volume is then monotonous and increasing with time. The dominance rule is the following: label l_1 dominates label l_2 if l_1 has a lower cost and a lower fictitious volume than l_2. A second difficulty is that this dominance rule can only be applied in the case where l_1 and l_2 have already turbined enough water to meet the minimum volume on future time periods. Indeed, in the case where the minimum volume is not reached, then a partial solution corresponding to less resource and better value at a time period t may be required to turbine on very uneconomic time periods between t and T to satisfy the lower bound. The algorithm defined allows to obtain the Pareto-optimal paths, considering the volume and the cost as the two criteria to be minimized. The authors stated that the number of partial solutions can be very large as long as the target volume is not reached.

There are also other HUC related problems solved by dynamic programming. In [4] the Hydro Unit Load Dispatch problem (HULD) is presented, where given a quantity of water, the goal is to optimize the distribution of this water through the different turbines. A dynamic programming algorithm and population-based optimization tools are presented. However this problem does not have any of the additional constraints of ramping and of min-up/down.

2.2 Shortest path with resource constraint

As the HUC can be seen as a shortest path problem with a resource bounded both from below and above, we are interested in the solution methods for the RCSPP (Resource Constraint Shortest Path Problem).

There are works on the RCSPP to solve the thermal problem on EDF instances [8]. In that paper it is indicated that the resource has an upper bound but no lower bound. However, as specified in [1], the difficulty of HUC comes from the lower bound on the volume, which prevents the use of dominance rules.

In the survey [17] a state of the art for different shortest path variants is described. More specifically, it is indicated that there is little work on the RCSPP with equality constraints, or window constraints (as is the case with the HUC). Three papers are cited, namely [14] with a heuristic, [3] with an integer formulation and [21] with a dynamic programming algorithm. As we look for an exact algorithm, we will focus on the three-phase algorithm described in [21]. This algorithm considers a given acyclic graph, where for each node there is an upper and a lower bound on the resource. The first step is to improve these bounds, so as to keep only intervals that can lead to a feasible solution. The second and third steps are the two steps of the algorithm described for the RCSPP in a subsequent paper from the same authors [20]. The second step lies in extending the graph, so that if there are two different paths to a node, then two nodes will exist in the new graph, one for each possible path. The extended graph becomes substantially similar to the graph of Figure 1(a). The problem is then solved by dynamic programming, in pseudo-polynomial time. This method seems to be efficient in particular as for the described instances, the resource is an integer which seems to be relatively small from the size of the extended graphs.

capacity C			0	1	2	3	4	5	6	7	8	9	10
n	w_n	v_n											
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	4	0	0	4	4	4	4	4	4	4	4	4
2	4	6	0	0	4	4	6	6	10	10	10	10	10
3	3	5	0	0	4	5	6	9	10	11	11	15	15
4	5	6	0	0	4	5	6	9	10	11	11	15	15
5	6	10	0	0	4	5	6	9	10	11	14	15	16

Table 1: Example of the knaspack dynamic programming algorithm

2.3 Unit Commitment dynamic programming

A dynamic programming algorithm for a single Unit Commitment (1-UC) with ramp and min up/down constraints is presented in [7]. The difference between the 1-UC and the 1-HUC is that there is no resource in the former. The dynamic programming algorithm presented is based on a graph with a source vertex and several groups of T vertices. For each odd group, node t indicates that the unit is turned on at time period t. For each even group, node t indicates that the unit is turned off at time period t. The arcs connect the vertices of a group to the next groups, from a time period t to a time period $t' > t$. Finding a path in this graph allows one to find an on-off schedule for the unit.

3 Dynamic programming for the simplified 1-HUC

The simplified 1-HUC is the core structure of the 1-HUC, i.e., without the ramp constraints and the min-up/down constraints. The constraints taken into account are (a) to (d).

3.1 Basic dynamic programming for the knapsack problem

There exists a classical dynamic programming algorithm for the binary knapsack problem, of pseudo-polynomial complexity $\mathcal{O}(nC)$, with n the number of variables and C the capacity of the knapsack. Let $KP(n, C)$ denote the knapsack problem with n variables and capacity C. The idea of this algorithm is to find the optimum of $KP(n, C)$ from either $KP(n-1, C)$ without selecting item n or $KP(n-1, C-w_n)$ by selecting the item n. We can then make a two-dimensional table of size $(n+1) \times (C+1)$, where cell $(n", C')$ contains the value of the optimal solution of $KP(n" - 1, C' - 1)$. We fill the first row with zeros, because the considered knapsacks do not contain any item. Once the table is filled using the dynamic programming algorithm, we can start from cell (n, C), and backtrack up to the first row of the table as illustrated in the following example.

Example : Let $KP(5, 10)$, the weights $w = [2, 4, 3, 5, 6]$ and the values $v = [4, 6, 5, 6, 10]$. **Table 1** is obtained with the dynamic programming algorithm. Starting from the cell $n' = 5$, $C' = 10$, the dotted arrows represent the computed cells. The bold arrows represent the path to obtain the optimal solution.

To adapt this algorithm to the case of the 1-HUC, we need to take into account the order constraints, the knapsack constraints, the inverted knapsack constraints and items with negative value.

3.2 Incorporating order constraints

Let us denote $PKP(n, C)$ the knapsack problem with order constraints, with n items distributed in groups and a capacity C. Order constraints are such that each item has at most one successor and at most one predecessor, as for the 1-HUC. Let $H(i)$ be the chain up to the item i. An idea to adapt the propagation rules is the following: The optimum of $PKP(n, C)$ is either the optimum of $PKP(n-1, C)$, without selecting the item n or the optimum of $PKP(n-|H(n)|, C-\sum_{n' \in H(n)} w_{n'})$, by selecting the item n. Thus, selecting item n implies selecting all the items in its chain, so we cannot have an invalid solution. We notice that for a chain $H(i)$, there are exactly $|H(i)|+1$ possibilities to select j items, with $j \in [0; i]$. Now, there is only one possibility to select j items, and that is to select the first j items of the chain. We could then reduce the number of lines to one line for each chain. There would be now $|H|+1$ arcs corresponding to choose $j \in [0, |H|]$ items of the chain, rather than 2 arcs per line. In the case of the 1-HUC, a chain corresponds to a time period.

Example : Let us take the case of a knapsack with order constraints as in the 1-HUC. Consider an instance with 2 groups of 3 items, ignoring weights and values for simplicity purpose. **Table 2b** (**Table 2a**, resp.) displays the arcs of the second group, (without, resp.) taking into account the order constraints There are four paths in **Table 2a**, whereas there are three in **Table 2b**, thus highlighting that there is one infeasible path with respect to the order constraints in **Table 2a**. Note that the number of feasible paths corresponds to the number of items in the second group, and that the paths never cross. We can then aggregate the lines of the same group to obtain **Table 2c**.

3.3 Incorporating nested (inverted) knapsack constraints

The bounds α_t and β_t from inequalities (a) and (b) can directly be taken into account by pruning infeasible nodes in the dynamic programming scheme. However, tightening these bounds may lead to more efficient schemes. First note that α_t and β_t are not bounds on the flow beginning at time period t, they are bounds on the sum of the flows from time period 1 to t inclusive. If the gap between β_t and α_t is large, then the dynamic programming will develop a large number of arcs, sometimes leading to infeasible solutions. For example, in the case of the HUC, it is very frequent to have α_t bounds very large compared to the flows, and negative β_t bounds. However, as the flow rates (weights for a knapsack) are positive, we know that a lower bound of 0 and an upper bound equal to the maximum flow rate will never be violated by a feasible solution. We can therefore introduce the bounds α'_t and β'_t in the following way:

$$\alpha'_t = \min(\alpha_t, t\sum_{i=1}^{M} D_i)$$
$$\beta'_t = \max(\beta_t, 0)$$

By using bounds α'_t and β'_t, we can drastically reduce the possibilities, but there are still cases where dynamic programming can build arcs leading to infeasible solutions. Suppose that at time period t close to zero we have a target volume C_t. Suppose that $\alpha'_{t+1} < C_t$. Then if the dynamic programming starts from time period T, there could be an arc to a solution with less than C_t amount of water flowed at time period $t+1$. Such a solution would be infeasible. Indeed since flows are positive, it is impossible to flow C_t at time period t, and less than C_t at time period $t+1$.

C		0	1	2	3	4	5	6	7	8	9	10
t	n											
0	0											
1	1											
	2											
	3											
2	4											
	5											
	6											

(a) Dynamic programming without order

C		0	1	2	3	4	5	6	7	8	9	10
t	n											
0	0											
1	1											
	2											
	3											
2	4											
	5											
	6											

(b) Dynamic programming with order

C	0	1	2	3	4	5	6	7	8	9	10
t											
0											
1											
2											

(c) Dynamic programming with one line per group

Table 2: Dynamic programming example with and without order constraints

Following this logic, we can compute the bounds of a time period from the bounds of the other time periods, following the rules below.

- if $t > t'$ then the maximal total flow until time period t is at most $\alpha'_{t'} + (t - t') \sum_{i=1}^{M} D_i$
- if $t < t'$, then the maximal total flow until time period t is at most $\alpha'_{t'}$
- if $t > t'$ then the minimal total flow until time period t is at least $\beta'_{t'}$.
- if $t < t'$ then the minimal total flow until time period t is at least $\beta'_{t'} - (t' - t) \sum_{i=1}^{M} D_i$.

Let us define $\widetilde{\alpha}_t$ and $\widetilde{\beta}_t$ as follows:

$$\widetilde{\alpha}_t = \min(\quad \min_{t'<t}(\alpha'_{t'} + (t - t') \sum_{i=1}^{M} D_i) \quad , \quad \min_{t'>t}(\alpha'_{t'}) \quad)$$

$$\widetilde{\beta}_t = \max(\quad \max_{t'<t}(\beta'_{t'}) \quad , \quad \max_{t'>t}(\beta'_{t'} - (t' - t) \sum_{i=1}^{M} D_i) \quad)$$

Tight bounds α^*_t and β^*_t are calculated as follows:

$$\alpha^*_t = \min(\alpha'_t, \widetilde{\alpha}_t)$$
$$\beta^*_t = \max(\beta'_t, \widetilde{\beta}_t)$$

Example : Let us consider an instance with $T = 6$ time periods. This instance has a constant inflow of water into the upstream reservoir, increasing the α_t and β_t bounds over time. This instance has two target volumes: for $t = 3$ and $t = 6$. Let C_t be the target volume at time period t. Let $\overline{\alpha}$ be the maximum of α_t, $t \leq T$ and $\underline{\beta}$ the minimum of β_t, $t \leq T$. Suppose that $\overline{\beta} > D$, with D the maximum total water flow that can be delivered in 6 time periods and $\underline{\alpha} < 0$.

By applying the tighter bounds we can see that we drastically reduce the possibilities, thus the number of arcs potentially developed by dynamic programming, as illustrated in Table 3.

3.4 A* Algorithm for the simplified 1-HUC

The A* algorithm works in the following way. We define a structure for the nodes, containing the following information:

- f: value of the objective function of the partial solution for reaching the node from a source
- h: value of the optimistic heuristic on the partial solution for reaching a target from the node.

We initialize a list of open nodes, containing the starting point. Then iterating as long as the list is not empty, we select a node which maximizes $f + h$, called current node. If the current node is a target, we save it if the value f is better than the that of the best solution found. Otherwise, we delete the current node from the list of open nodes and add its neighbors to the list.

C	$\underline{\alpha}$			0		C_3			C_6			D			$\overline{\beta}$
t															
0															
1												X	X	X	X
2	X												X	X	X
3	X	X	X	X	X		X	X	X	X	X	X	X	X	X
4	X	X	X												X
5	X	X	X	X											
6	X	X	X	X	X	X	X	X		X	X	X	X	X	X

(a) Table with bounds α_t and β_t

C	$\underline{\alpha}$			0		C_3			C_6			D			$\overline{\beta}$
t															
0															
1	X	X	X			X	X	X	X	X	X	X	X	X	X
2	X	X	X				X	X	X	X	X	X	X	X	X
3	X	X	X	X	X		X	X	X	X	X	X	X	X	X
4	X	X	X									X	X	X	X
5	X	X	X	X									X	X	X
6	X	X	X	X	X	X	X	X		X	X	X	X	X	X

(b) Table with bounds α'_t and β'_t

C	$\underline{\alpha}$			0		C_3			C_6			D			$\overline{\beta}$
t															
0															
1	X	X	X			X	X	X	X	X	X	X	X	X	X
2	X	X	X				X	X	X	X	X	X	X	X	X
3	X	X	X	X	X		X	X	X	X	X	X	X	X	X
4	X	X	X	X	X				X	X	X	X	X	X	X
5	X	X	X	X	X	X				X	X	X	X	X	X
6	X	X	X	X	X	X	X	X		X	X	X	X	X	X

(c) Table with bounds α^*_t and β^*_t

Table 3: Reducing the dynamic programming search space using bounds on the flows

3.4.1 Optimistic heuristic

To use A* for our problem, we must define an optimistic heuristic, i.e., one that overestimates the value of the objective function because we are solving a maximization problem. To do this, we can calculate a linear relaxation, in several steps. **Step 1:** Start with a solution that contains no items. **Step 2:** Sort the items in descending order of profitability, i.e., w.r.t. ratio value/weight. **Step 3:** As long as there is a lower bound α_t not respected, look for the lower bound not respected for the smallest time period t. Look for the most profitable item for time periods at most t. If this item is of positive profitability, take it as much as possible provided the upper bounds are respected. If this item is of negative profitability, take it as little as possible to try to respect the lower and upper bound. **Step 4:** As long as there is an item of positive profitability, take it as much as possible provided the upper limits are verified.

The returned solution is a linear relaxation of our problem, because steps 3 and 4 do not necessarily select whole items. However we can slightly improve the returned solution. An integer solution is necessarily of a weight which is a combination of the weights of the items. Therefore the weight of an integer solution is necessarily a multiple of the greatest common divisor (GCD) of the weights of the items. When the heuristic selects an item in a fractional way, we can increase or reduce the fraction so that the weight of the returned solution remains a multiple of the GCD of the weights of the items. Note that since the weights are the same from one time period to another, we can quickly compute the GCD by considering only the weights of a single time period.

3.4.2 Reduction of the number of open nodes

We have bounds at each time period, so we can use them when we add nodes to the list of open nodes. Indeed, a node which does not respect one of these bounds cannot lead to a feasible solution, so it is not necessary to add it.

Suppose that two nodes represent a partial solution at time period t with the same total water flow. By induction, the partial node with the smallest value f cannot lead to an optimal solution, so it can be removed from the list.

The nodes in the list of open nodes are not necessarily for the same time period. It is possible that a node u of a time period t joins the path of a node v of a time period $t' > t$. If value f of node u is smaller than value f relative to the path of node v at time period t, then an optimal solution cannot be obtained by passing through u. If the converse is true, then we can construct a new node w, taking the path from u to time period t, then the path from v at time period $t + 1$ to t', and remove u and v from the list.

3.4.3 Implementation

The implementation is done in C++. A simple way to quickly find the current node from the list of open nodes is to have this list sorted. This list is subject to a lot of changes. Let us note n the size of the list. Regularly using a sorting algorithm of complexity $nlog(n)$ can become cumbersome, especially since this list can be relatively long. A more efficient way to keep this list sorted is to insert new nodes at the location that keeps the list sorted. As the list remains sorted, finding the location

can be done with a dichotomy of complexity $\log(n)$. Note that the same idea is used to sort the items to compute the heuristic.

3.5 Instances and results

A set of 8 instances is constructed, all of them are realistic and derived from EDF instances. Instances 1 to 3 have 4 operating points, instances 4 to 6 have 7 and instances 7 and 8 have 8. All instances have $T = 96$ time periods. After data conversion, the flow rates and powers are on the order of 10^3 to 10^4, with volumes on the order of 10^7. Instances 1, 4, and 7 have a binding maximum volume at the last time period. Instances 2 and 5 have a minimum binding volume at the last time period. Instances 3, 6 and 8 have a target volume at the last time period. Note that the target volumes are not equality constraints, they are maximum and minimum volumes that are very close, e.g. with a difference of 2000. In the case of an equality constraint, we often find an infeasibility which is instantly found by the MILP solver CPLEX.

To have a reference, we solve these 8 instances with CPLEX, version 12.8, on a single thread in default setting. A time limit of one hour is fixed. We also solve these 8 instances with the A* algorithm, on a single thread. **Table 5** gives the value of the objective, the gap and the computation time to solve these instances with both CPLEX and A* algorithm. If CPLEX proves optimality, the gap is noted "opt".

	CPLEX			A*	
instance	value	gap	time (s)	value	time (s)
1	-25434.5	opt	2832	-25434.5	**2**
2	43009	opt	470	43009	**2**
3	3550.82	opt	1167	3550.82	**2**
4	2457.41	opt	66	2457.41	**12**
5	111105	opt	64	111105	**33**
6	-1713.3	opt	509	-1713.3	**16**
7	5687.49	0.3%	3600	5687.49	**227**
8	16576.2	opt	**59**	16576.2	293
15	-71650.7	opt	**91**	-71650.7	191
16	-525454	opt	1385	-525454	**212**
17	44418.8	opt	939	44418.8	**277**
18	-20335.4	0.59%	3600	-20322.1	**733**
19	-103439	opt	**131**	-103439	136

Table 4: Performance of CPLEX and our algorithm on difficult instances for CPLEX

We notice that CPLEX reaches twice the maximum time of 3600 seconds, while our algorithm requires at most 733 seconds. The following time averages use the time limit of 3600 seconds when the instance is not solved, so for CPLEX the average time is an underestimate.

For CPLEX, the average resolution time is:

- 1145 seconds on the difficult instances

	CPLEX			A*	
instance	value	gap	time (s)	value	time (s)
21	-25436	opt	**0**	-25436	**0**
22	42991.1	opt	**0**	42991.1	**0**
23	3694.51	opt	**0**	3694.51	**0**
24	2457.41	opt	**2**	2457.41	4
25	11142	opt	**3**	111142	9
26	-1467.01	opt	**0**	-1467.01	2
27	5939.75	opt	**4**	5939.75	253
28	16725.4	opt	**0**	16725.4	15
35	-71505.4	opt	**0**	-71505.4	41
36	-525253	opt	**18**	-525253	43
37	44644.4	opt	**0**	44644.4	6
38	-20158.6	opt	**1**	-20158.6	150
39	-103343	opt	**0**	-103343	31

Table 5: Performance of CPLEX and our algorithm on easy instances for CPLEX

- 2 seconds on the easy instances
- 574 seconds on all instances

For our algorithm, the average resolution time is:

- 164 seconds on the difficult instances
- 42 seconds on the easy instances
- 103 seconds on all instances

4 Methods for solving the complete 1-HUC

The full 1-HUC takes into account the ramping and min-up/down constraints. The constraints considered are (a) to (h).

4.1 Extension of the A* algorithm to ramping and min-up/down constraints

On the one hand, these additional constraints seem to be advantageous for CPLEX. The reason is that these new constraints can break symmetries: if a very profitable time period t is between two unprofitable time periods $t-1$ and $t+1$, then the linear relaxation will not consider turbining at t as a possibility for an optimal solution, because it would be necessary to turbine on one of the unprofitable time periods. It will then be preferable to turbine on successive profitable time periods.

On the other hand, these constraints seem to be disadvantageous for the A* algorithm. First, these constraints make us lose several of the dominance properties between two solutions with the

same total water flow at the same time period. Indeed, we must now verify that two solutions with both the same total water flow, and the same time period can get access to the same operating points at the next time period. If for two partial paths all these characteristics are identical, then we can apply dominance properties. Second, a greedy algorithm with continuous variables as used previously does not scale well with the additional constraints, and will not necessarily produce an optimistic solution. The heuristic used is the greedy heuristic with continuous variables without the additional constraints, but this heuristic gives poor quality bounds. In the end the A* algorithm is not efficient for the full 1-HUC.

4.2 A new algorithm based on a bi-criteria approach

A new idea is to consider the volume at the last time period as an objective function, which leads to a bi-criteria problem.

We adapt the graph shown in **Figure 1**(b) to incorporate the additional constraints. To reduce the number of turbine on-off cycles, min-up and min-down constraints with a duration of $L_i = l_i = 2, i \leq M$, time periods are considered in realistic HUC instances. We then define a graph, where for each time period we have $2M$ nodes, indicating the operating point of the time period t, as well as if this operating point is higher or lower than time period $t-1$. In this way, we build only valid arcs, satisfying the constraints of ramping and min up/down on 2 time periods. In total this graph contains at most $T.2M$ nodes.

With the volume as a criterion, we come up with a two-criteria shortest path problem. The idea is to adapt a two-phase method, initially introduced for the two-criteria knapsack problem [18]. This classical method is particularly efficient in bi-criteria (or even tri-criteria) when the mono-criterion problem is easy to solve. In the bi-criteria case, the two-phase method is as follows. The first step is to aggregate the bi-criteria objective function into a mono-criteria objective function, using a convex combination. By varying the parameters of the convex combination, we can generate all the Pareto-supported solutions, i.e., the solutions that form the convex envelope of the solutions, in the sense of the optimization. Knowing these solutions, the space of Pareto-optimal solutions is drastically reduced. The second step is to use an enumeration algorithm, e.g. K-best or Branch and Bound, to obtain the remaining Pareto-optimal solutions.

The rationale behind adapting the two-step method is that the Pareto-optimal solutions are by definition not comparable, while in our case they can be. The thing is that we want to maximize only the value, not the volume used. Therefore, the first step is not to generate all the supported solutions, but only the ones that are close to the solution maximizing the value, while respecting the target volume constraints. For the second step of the algorithm, we rely on the solutions found in the first step to generate only solutions in a reduced subspace until generating the optimal solution.

We can see that depending on the instance, the Pareto front can have two forms. The first form occurs when it is not profitable to turbine: then the Pareto front will represent trade-offs between the volume and the objective function. The second form occurs when it is profitable to turbine, then the Pareto front will be represented by a single solution, the one where the maximum has been turbined. In the latter case, we will consider the volume as a cost. Thus, the more one turbines, the higher the cost associated with the volume. Therefore turbining a lot or very little produces two incomparable solutions forming a Pareto front. In the remainder of this section, the first case will be used and the

second case will be referred to as the case of an "inverted instance". All the procedures are adapted to handle both cases.

4.2.1 First phase

For the first phase, one difficulty is due to the volume and the value of the objective function not having the same scale. However, we do not want to change the scale of the volume, because the HUC is a problem very sensitive to the order of magnitude used for the volume in realistic instances [15]. When using MILP solvers – like CPLEX – based on floating-point arithmetic, changing the volumes scale can lead to numerical errors, thus leading to either infeasibility or loss of accuracy. To overcome this scale disparity issue, we introduce two coefficients, δ_V and δ_f which will be respectively the coefficient assigned to the volume and the coefficient assigned to the objective function value. These coefficients are used to aggregate the two criteria into a mono-criterion function. The goal of the first phase is to compute values of δ_V and δ_f such that two neighboring supported solutions have the same value for the aggregated function. Moreover, one of these solutions must not satisfy the minimum volume constraints while the other must satisfy them. In the case of an inverted instance, we will apply the same reasoning with the maximum volume. Thus, when we enumerate solutions in the second phase with coefficients δ_V and δ_f, the enumerated solutions will be close to the optimal solution.

To calculate δ_V and δ_f, the procedure is as follows. We compute a shortest path solution maximizing the volume ($\delta_V = 1$ and $\delta_f = 0$), and a solution maximizing the objective function value ($\delta_V = 0$ and $\delta_f = 1$). Then, we keep $\delta_f = 1$, and we compute δ_V such that these two solutions have the same value for the aggregate function. We recalculate a shortest path with these values. We iterate until the new solution found is one of the two solutions already obtained. In the case of an inverted instance, we just have to change the sign of δ_V so that the volume becomes a cost in the aggregate function.

We have some first promising results for this method. **Table 6** shows the computation time of step 1, as well as the values of 2 different solutions. SINF (resp. SSUP) represents the solution with the worse (resp. best) value of the two solutions.

Note: The generated solutions are not necessarily valid. Indeed, there are two cases for a generated solution to be invalid: either it does not satisfy a target volume of the last time period, or it does not satisfy one of the maximum or minimum volume on a time period different from the last time period. However, it is possible to know in advance when we are guaranteed to obtain a feasible solution. Indeed, one of the solutions is feasible if all the volume bounds, except the lower bound on the volume at the last time period, are respected by any solution. Since we know that one of the solutions satisfies the lower bound on the volume at the last time period, it necessarily respects all the bounds on the volume. In the inverted case, the same applies to the upper bound on the volume at the last time period. If this condition is not verified, then it is not guaranteed that any of the returned solutions is feasible. This condition is easily checked and verifiable upfront: it is enough to check if turbining at the maximum at each time period, or turbining at the minimum at each time period, allows for violation of one of the constraints on the volume.

instance	time (s)	SINF	SSUP	gap(%)
1	0.014	-31243.9	-25785.5	21.2
2	0.017	42321.9	42933.5	1.4
3	0.019	1759.74	3809.39	53.8
4	0.026	-1036.31	1748.99	159.2
5	0.034	107997	108244	0.2
6	0.028	-4257.77	-1786.71	138.3
7	0.042	2703.96	4610.37	41.3
8	0.046	9518.06	15048.3	36.7
15	0.063	-88211.3	-68376.3	29.0
16	0.065	-644199	-600284	7.3
17	0.077	41292.6	42171.9	2.1
18	0.058	-47615.8	-31083.5	53.2
19	0.081	-194428	-185439	4.8

Table 6: Time and value of the solutions obtained by the first step of the algorithm

4.2.2 Second phase

In bi-criteria optimization, this second phase is applied when all pairs of solutions from the first phase produce a triangle in the solution space, in which all Pareto-optimal solutions are found. However, we have a different case from the classical bi-criteria for several reasons. First, it is possible that both solutions returned by the first step are infeasible solutions for the 1-HUC. Thus, it is possible that no 1-HUC feasible solution is in the triangle formed by these two solutions, or even that no 1-HUC feasible solution is Pareto-optimal in bi-criteria. Second, we are only looking for a single optimal solution and, consequently, we will only focus on the solution space around the two solutions returned by the first step.

There are several options for solving the second phase of the two-phase algorithm. We first enumerate the different possibilities, and then we present the implemented approach.

Pareto front enumeration A common method is a Pareto front enumeration approach, for which there exists different algorithms. The reasons for preferring this method are as follows. First, we only look for a solution in a particular area, which should be smaller in our case than in the classical case. Second, by directing the enumeration, we can expect to quickly generate Pareto-dominated solutions in the search area. This would be interesting in the case where no solution exists in the restricted area defined by the two infeasible solutions from the first step. Compared to the classical case, we would have a compromise: on the one hand, we are looking for a potentially Pareto-dominated solution, therefore in a larger area than one of the triangles of the classical case; on the other hand we have only one search area, contrary to the classical case where we have several triangles.

Nonlinear metric A second method would be to consider a nonlinear metric, like the Chebychev norm [13], the Choquet integral or the Lorenz dominance. A nonlinear metric can be more computationally demanding, and should not be a priority as long as there are other good options that could

be obtained more efficiently. Moreover, an optimal 1-HUC solution is not necessarily Pareto-optimal, we should check if these algorithms can be adapted to our case.

Interactive algorithm Another classical method is to use an interactive algorithm. This type of algorithm will produce a Pareto-optimal solution, and then ask a decision-maker his preferences towards this solution: if the solution suits him or if the solution favors/disfavors too much one of the criteria. The algorithm will then produce a new solution by taking into account the preferences of the decision maker. In our case, we could automate the responses of the decision maker because for any instance we want to maximize the value while respecting the limits on the volume. The problem with an interactive method is that we have to add at each iteration bounds on the values of one or more objectives. Adding these constraints would bring us back to the case of an upper and/or lower constrained shortest path, which can become difficult to solve.

Robust shortest path Another approach could be to consider the robust shortest path, considering the volume and the objective function value as objective function values in two different scenarios. There are pseudo polynomial algorithms for two robust shortest path problems [19]

- The shortest path minimizing the objective in the worst case scenario (thus minimizing the maximum of the criteria if we see the scenarios as criteria)
- The shortest path minimizing the maximal deviation in the worst case scenario to the objective of a given target solution.

In both cases, the algorithm is pseudo-polynomial because it is linear in the size of the objective of the scenarios. However, we know that for the HUC the volume and the value of the objective function can be very large without being integer, therefore this method could be ineffective.

Scalarization Another idea would be to scalarize the volume and value of the objective function. Let A and B be the two solutions obtained by the first phase, with A.volume<B.volume and A.value>B.value. We could scalarize the volume and the value such that A.volume=B.value. We would then try to find a solution C that maximizes min(C.volume,C.value). According to this metric, point A and B have the same value, and any point of the triangle constructed by A and B would have a better value. We would then replace A or B by C and we could iterate until convergence. However we see two difficulties: maximizing the worst criterion does not seem to be solvable by dynamic programming, thus requiring the use of a solver, and scalarizing the volume can lead to floating point computation errors on the volumes [15].

Heuristics One could also consider heuristics:

- Local Branching to obtain solutions close in term of decision to those of the phase 1 but which would be inside the triangle containing the optimal.
- Repair a solution in the same way as CPLEX when given an infeasible MIP start.
- Approximate enumeration of the Pareto front

Pulse algorithm For numerical experiments, a first approach has been tested using the Pulse algorithm described in [5]. This algorithm generates the Pareto front of a bi-criteria shortest path problem. The idea is to use a DFS to traverse the feasible solutions, and use multiple checks to remove subpaths that cannot lead to a Pareto-optimal path. In the literature, this algorithm obtains the Pareto-optimal front in a few seconds for graphs with 300,000 nodes and 1,000,000 arcs. As our instances have at most 40 000 nodes, this algorithm would seem to be efficient. Moreover, we do not want to generate the complete Pareto front, we want at most to generate the Pareto front in the triangle obtained by the 1st phase, and at least one unique solution of this triangle that would maximize the value of the objective function while respecting the volumes. In our case, despite the information given by the two points calculated in the first phase, the paths start to be eliminated in almost all cases when we are at time period 80 or more, which leads to a combinatorial explosion before partial solutions could be pruned.

The K-best solutions algorithm of [6] The K-best solutions algorithm is another possible solution enumeration algorithm. An algorithm for digraphs , of complexity $\mathcal{O}(m + nlogn + k)$ has been proposed by [6]. The idea is to solve the shortest path. From the obtained solution, for each arc of the solution, this arc is removed from the initial graph, and then the problem is solved to obtain an optimal solution on this new graph. The same process is repeated for the new solutions, starting with the best solutions obtained, until the K best solutions are obtained. We implemented this algorithm, but, because of the particular structure of our graph (2 nodes per operating point), it produces a large number of similar solutions, and is not efficient.

Selected approach We then propose a new algorithm, which presents some similarities to the K-best solutions algorithm of [9] with complexity $\mathcal{O}(Kn^3)$, where n the number of nodes in the graph. Contrary to the bi-criteria methods, the search area has not necessarily the form of a triangle. Indeed, as indicated previously, it is possible that the two returned solutions are not valid, and that no feasible solution is located in the triangle formed by these solutions. Figure 2 illustrates a case where the triangle would not contain any feasible solution. As long as no feasible solution is found, the search area will be defined by a semi-infinite polygon. The figure on the top of Figure 3 describes a search space bounded by: the maximum volume at the last time period; the minimum volume at the last time period, and the tangent passing through the two solutions of the first phase. This space is not bounded on the left, because we do not know a minimum value for the feasible solutions. If a feasible solution is found, we will define two spaces: a semi-infinite rectangle and a polygon. The figure on the bottom of Figure 3 describes these two search spaces. The polygon is bounded in this case by: the minimum volume at the last time period; the value of the feasible solution found, and the tangent passing through the two points of the first phase. Note that if the maximum volume is very low, then the top of this triangle would be cut off by the bound on the volume, hence this area is a polygon, not necessarily a triangle. The rectangle is bounded by: the tangent passing through the two solutions of the first phase, and the tangent passing through the nadir point of the polygon. The nadir point of the polygon is the point of the polygon being worse or equivalent, on all criteria, than the other points of the polygon. The reason for using two zones is because we will rely on the rectangle to remove partial solutions, and on the polygon/triangle to remove complete solutions.

The algorithm is as follows. First, the coefficients δ_V and δ_f are retrieved to obtain the optimization direction, as well as the feasible solution, if any. The longest paths are no longer computed in preprocessing, but are computed and stored only if they are necessarily during enumeration. In the worst case, the longest path is computed for each node, and in practice it is necessary to compute the longest path only for a small set of nodes, which reduces the total computing time. Note that these longest paths are optimistic heuristics, because we do not take into account the bounds on the volume on time periods different from T. Third, we create a list $\mathcal{L}$, which will contain solutions that we will call hybrid (as explained in the algorithm). This list is initialized with the shortest path from the source to the last time period and the algorithm proceeds as follows:

- The first hybrid solution of $\mathcal{L}$ is removed. This solution is noted $S = (u_1, u_2, ..., u_T)$ with u_t the node of time period t.
- We assume without loss of generality that S was obtained by fixing the first k decisions: $u_1, ..., u_k$ ($k = 0$ at initialization). For any $t > k$ and $v_t \neq u_t$ such that (u_{t-1}, v_t) is an arc of the graph, we proceed with the following loop:
  - Create a new fixed path: $u_1, ..., u_{t-1}, v_t$.
  - Check that we have not already created a fixed path to v_t with the same volume and a larger value (in which case we will not be able to generate the optimal solution).
  - Check that the new fixed path satisfies the volumes from time period 1 to t.
  - Check that the volume turbined by the fixed path is still consistent with the bounds on the volume at the last time period.
  - Form a new hybrid solution with the created fixed path complemented by the longest path of v_t until the last time period (we compute it if it has not already been computed).
  - If the new hybrid solution is feasible and with a better value than the feasible solution found, i.e., in the polygon/triangle, then update the two search areas with this new solution.
  - Otherwise, if the solution is not completely fixed and is with a better value than the nadir point of the search area, then add it to $\mathcal{L}$, keeping $\mathcal{L}$ sorted by decreasing solution value.
  - Remove from $\mathcal{L}$ any hybrid solution whose value is smaller than the nadir point of the search area, i.e., which are not in the rectangle.
- Stop if $\mathcal{L}$ is empty.

In the case of an inverted instance, the procedure is identical, except that since δ_V is negative, the volume becomes a cost. Then, the direction of optimization and the search areas will be reversed horizontally in **Figure 3**.

We underline the following differences between our algorithm and that of [9]. We compute longer paths in preprocessing, which is a direct improvement of the algorithm because in the case of K-better, we go from a complexity $\mathcal{O}(Kn^3)$ to $\mathcal{O}(n^3 + K)$. The algorithm stops when it has converged (K is not known in advance). We have specific constraints to satisfy, namely the bounds on the volume at each time period. As we are at the frontier between the mono-criteria and the bi-criteria, we have additional steps that take into account the management of the triangle containing the optimum and

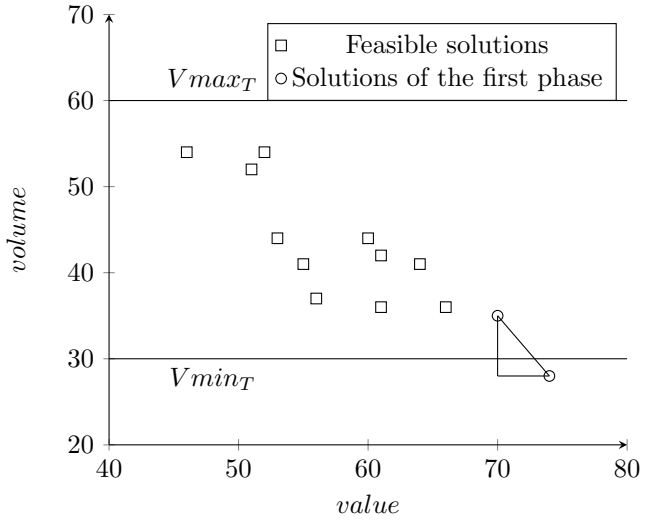


Figure 2: Pathological case to use a triangle

the list of the paths fixed pareto-optimum for each node. When a solution is fixed, in the algorithm of [9], only the fact of not selecting u_t is fixed, whereas in our algorithm we create a new fixed path for all v_t accessible from u_{t-1}.

5 Numerical results

This section describes an experimental comparison between CPLEX and the proposed two-phase algorithm. The version of CPLEX used is 12.8, and in both cases with a single thread. The first phase of the algorithm, to generate the first triangle, is not shown because as can be seen in Table 6, this step takes less than 0.1 seconds for each instance. Five sets of instances are used :

- A first set containing 13 instances derived from EDF instances with 96 time periods and between 4 and 22 operating points.
- A second set of instances containing 105 instances, generated with the Hydro Instance Generator (HIG) provided online by Dimitri Thomopulos (HIG[1]), with 48 time periods, and 20 operating points. The prices and inputs are derived from the Italian market. This instance set contains one instance per month, between April 2004 and December 2012, thus allowing for diversification.
- A third instance set containing 105 instances built from the HIG instances. The modification made is on the prices: for each time period, we randomly choose the price Λ_t in the interval $[0.95\Lambda_1; 1.05\Lambda_1]$.
- A fourth set of instances containing 70 EDF instances, with 96 time periods, and 3 to 10 operating points.

[1]https://people.unipi.it/dimitri_thomopulos/libraries/hig/

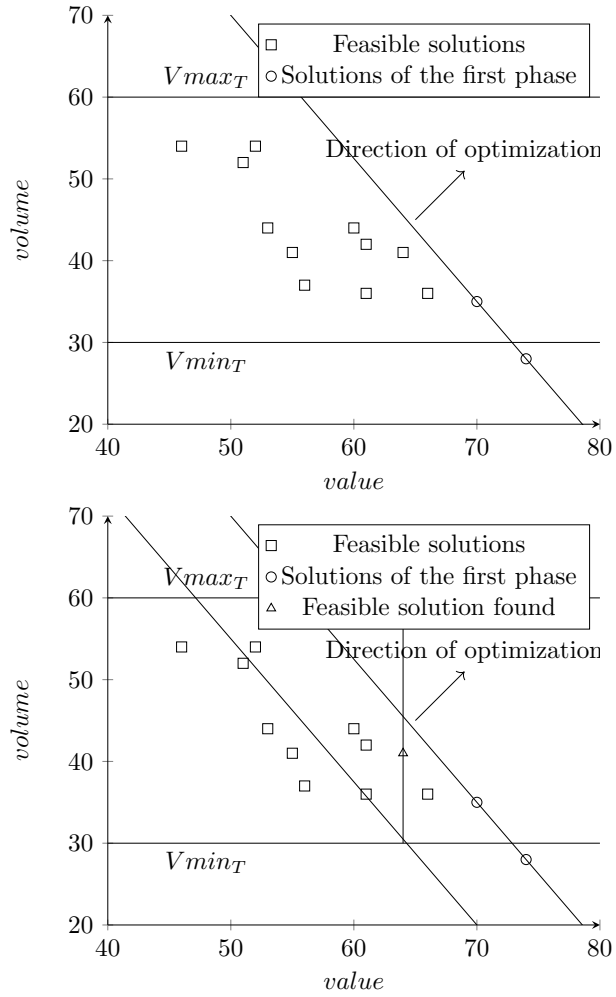


Figure 3: Representation of the solution search space during the algorithm

- A fifth set of instances, containing the same 70 EDF instances, with prices from one iteration of the Lagrangian decomposition[2].

On the EDF instance sets 1 and 4, the prices have been generated randomly as for HIG instance set 3.

Table 7 compares for each instance set the performance of CPLEX and the proposed two-phase algorithm in terms of the number of instances solved to optimality with a time limit of 3600 seconds. For each set and each algorithm Table 7 provides the number of unsolved instances (#fail), the average time on solved instances (avg-t-res) and the average time on all instances (avg-t) where unsolved instances count for 3600 seconds.

Table 7 reveals that instance set 3, that we built by modifying the prices used in the HIG generator instances, is particularly difficult for CPLEX, with a failure rate of almost 30% and an average solving time of 97.4 s for solved instances, while for our algorithm the failure rate is almost 5% with an average CPU time of 167.7 s for solved instances. We can then assume that instances with similar prices on the time periods will tend to be more difficult for CPLEX. This phenomenon can be explained by a large number of symmetric fractional solutions with nearly identical values. CPLEX cuts will eliminate some symmetries, but not all, thus leading to a large number of branches.

The detailed results are given in Table 8 for instance set 1, Tables 9–17 for instance set 2, Tables 18–26 for instance set 3, Tables 27–28 for instance set 4 and Tables 29–30 for instance set 5. For each instance the tables give the objective function value (value) obtained by CPLEX and by the two-phase algorithm. They also provide the number of iterations (#iter) and the CPU time (time) of the second phase for the two-phase algorithm as well as the final gap (gap), the CPU time (time) and the number of nodes (node) for CPLEX.

For EDF instance set 1 (Table 8), the results show that CPLEX has an unstable behavior: on some instances it solves the problem almost instantaneously, while non negligible CPU times are observed on instances 1, 3, 15 and 18 (from 15 to 95 seconds) and even very large CPU times on instances 16 and 19 (2045 seconds and time limit reached with a 1.4% gap, respectively). In comparison, the two-phase algorithm has a much more stable behavior with CPU time (from 0.83 to 15.5 seconds) and outperforms CPLEX on these 6 instances.

For instance set 2 generated with the HIG generator (Tables 9–17), CPLEX solves all instances under 5 seconds with a large majority of instances being solved in a negligible CPU time and a few branch and bound nodes. The two-phase algorithm performs reasonably well as a majority of instances are solve under 2 seconds but in some cases the required CPU time is much larger (up to 40 seconds for instance 20040505).

As already mentioned, for instance set 3 corresponding to HIG instances with modified prices, CPLEX reaches 30 times the CPU time limit with a strictly positive gap out of the 105 instances and has an important CPU time for 9 other instances (from 11 to 1728 seconds). On all these instances, our algorithm is generally much faster.

For EDF instance sets 4 and 5 with 3 to 10 operating points, CPLEX performs extremely well, independently on the prices. In comparison, our algorithm is slower on some more difficult to solve instances where the time limit is reached.

[2]At EDF the prices of the HUC come from a Lagrangian decomposition of the global unit commitment problem including all production unit types

Note that a hybrid algorithm has also been tested where first CPLEX's heuristic finds a feasible solution, then the latter is used for the two-phase algorithm as a warm-start. However, the results in terms of number of iterations and computational times are exactly the same with or without warm-start.

6 Conclusion

In this paper, we have proposed an efficient A* algorithm to solve a simplified variant of the single unit hydro unit commitment problem incorporating lower and upper bounding constraints on the cumulative water flow. The algorithm is shown to have a more stable behavior than CPLEX on realistic EDF instances.

For the complete variant of the 1-HUC including both the ramping and min-up/min-down constraints the A* algorithm as well as the standard resource-constrained shortest path approaches are shown to be ineffective. Such a change is induced by the weakening of the dominance rules.

To overcome this difficulty, we have proposed a two-phase algorithm inspired by the algorithms used in bi-criteria optimization. The first phase solves extremely fast simple shortest path problems while the second phase is derived from a K-best solutions algorithm to enumerate feasible solutions in a restricted search space.

We have compared this algorithm with the MILP solution approach using CPLEX on several sets of instances. We have exhibited instances with a small range of price variation, that are very challenging for CPLEX. For these instances, the two-phase algorithm is generally more efficient. In the case of a larger range of variation, CPLEX solves the instances almost instantaneously, whereas the algorithm in two phases takes a few seconds. These results show a better robustness in terms of performance of the two-phase algorithm compared to CPLEX.

One may argue that instance set 3 with a small price variability is artificially too hard. In practice, the values are not nearly identical throughout the time horizon. One could imagine a more realistic case with three subsets of time periods: time periods with high prices in the peak hours, time periods with low prices in the off-peak hours, and transient time periods with medium prices. In this kind of setup, the number of time periods is large enough for the the number of symmetric fractional points to increase dramatically.

As the two approaches appear to be complementary on most instances, a practical approach would be to launch CPLEX for a few second and, in case optimality has not been proven, launch the two-phase algorithm.

An interesting perspective would be to include the proposed algorithms in the subproblem of decomposition schemes to solve the HUC on valleys with several units.

	Two-phase algorithm			CPLEX		
instance set	#fail	avg-t-res (s)	avg-t (s)	#fail	avg-t-res (s)	avg-t (s)
1	0	3.0	3.0	1	187.4	449.8
2	0	2.1	2.1	0	0.1	0.1
3	5	167.7	331.2	30	97.1	1094.9
4	1	0.1	51.6	0	0.0	0.0
5	2	0.4	103.3	0	0.1	0.1

Table 7: Performance of CPLEX and the two-phase algorithm on the 5 sets of instances

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
1	-27098.1	2573	0.83	-27098.1	opt	15.0	34237
2	42332.8	2830	0.91	42332.8	opt	1.0	8434
3	2474.97	6285	2.05	2474.97	opt	65.0	68998
4	1551.19	358	0.59	1551.19	opt	0.0	660
5	108119.0	858	1.0	108119.0	opt	0.0	842
6	-2557.29	366	0.58	-2557.29	opt	0.0	630
7	3025.79	4759	2.38	3025.79	opt	0.0	110
8	13898.3	15264	15.52	13898.3	opt	0.0	256
15	-81463.8	6459	5.59	-81463.8	opt	28.0	11017
16	-622053.0	909	2.35	-622053.0	opt	2045.0	110200
17	41734.1	24	1.61	41734.1	opt	0.0	0
18	-32135.6	596	2.98	-32135.6	opt	95.0	14163
19	-185788.0	311	2.27	-185788.0	1.4%	3599.0	187073

Table 8: Second phase of the two-phase algorithm and CPLEX comparison for EDF inspired instances

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20040405	12517.5	2264	1.91	12517.5	opt	0.0	162
20040505	29528.8	35714	40.86	29528.8	opt	5.0	12305
20040605	6387.36	283	1.41	6387.36	opt	0.0	0
20040705	3300.02	12	1.35	3300.02	opt	0.0	0
20040805	2059.17	14	1.26	2059.17	opt	0.0	0
20040905	171.11	27	0.23	171.11	opt	0.0	0
20041005	150.2	6	0.09	150.2	opt	0.0	0
20041105	2305.81	64	1.34	2305.81	opt	0.0	2
20041205	33048.0	1301	2.58	33048.0	opt	0.0	0

Table 9: Second phase of the two-phase algorithm and CPLEX comparison for the year 2004

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20050105	7257.47	141	1.39	7257.47	opt	0.0	8
20050205	3444.13	166	1.35	3444.13	opt	0.0	39
20050305	2990.66	71	1.37	2990.66	opt	0.0	0
20050405	3065.66	22	1.34	3065.66	opt	0.0	0
20050505	8014.0	293	1.43	8014.0	opt	0.0	42
20050605	3682.13	14	1.31	3682.13	opt	0.0	0
20050705	4152.13	103	1.36	4152.13	opt	0.0	9
20050805	435.4	47	0.37	435.4	opt	0.0	0
20050905	128.75	14	0.09	128.75	opt	0.0	0
20051005	15050.9	124	1.27	15050.9	opt	0.0	0
20051105	55498.5	1793	2.16	55498.5	opt	0.0	0
20051205	17538.7	521	1.52	17538.7	opt	0.0	0

Table 10: Second phase of the two-phase algorithm and CPLEX comparison for the year 2005

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20060105	18455.6	2222	2.33	18455.6	opt	0.0	138
20060205	12605.8	553	1.5	12605.8	opt	0.0	62
20060305	23526.5	525	1.65	23526.5	opt	0.0	11
20060405	14607.8	422	1.59	14607.8	opt	0.0	10
20060505	6071.94	18	1.37	6071.94	opt	0.0	0
20060605	3205.81	39	1.22	3205.81	opt	0.0	0
20060705	1103.54	59	0.44	1103.54	opt	0.0	0
20060805	1146.48	75	0.66	1146.48	opt	0.0	0
20060905	0.0	1	0.02	0.0	opt	0.0	0
20061005	211.18	10	0.15	211.18	opt	0.0	0
20061105	1819.43	44	0.78	1819.43	opt	0.0	0
20061205	2144.53	85	0.74	2144.53	opt	0.0	0

Table 11: Second phase of the two-phase algorithm and CPLEX comparison for the year 2006

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20070105	3090.91	23	1.21	3090.91	opt	0.0	0
20070205	2931.72	59	1.0	2931.72	opt	0.0	0
20070305	2953.03	34	1.12	2953.03	opt	0.0	0
20070405	2828.1	26	1.32	2828.1	opt	0.0	0
20070505	18005.1	253	1.34	18005.1	opt	0.0	0
20070605	22707.1	352	1.37	22707.1	opt	0.0	57
20070705	3541.29	19	0.99	3541.29	opt	0.0	0
20070805	816.28	64	0.51	816.28	opt	0.0	0
20070905	2702.08	93	0.87	2702.08	opt	0.0	0
20071005	797.0	29	0.36	797.0	opt	0.0	0
20071105	0.0	1	0.02	0.0	opt	0.0	0
20071205	965.6	36	0.44	965.6	opt	0.0	0

Table 12: Second phase of the two-phase algorithm and CPLEX comparison for the year 2007

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20080105	13811.4	87	1.34	13811.4	opt	0.0	5
20080205	29256.8	528	1.42	29256.8	opt	0.0	17
20080305	17781.6	117	1.36	17781.6	opt	0.0	0
20080405	3115.64	13	1.36	3115.64	opt	0.0	0
20080505	4184.53	18	1.32	4184.53	opt	0.0	0
20080605	13380.2	145	1.4	13380.2	opt	0.0	40
20080705	12684.5	11	1.27	12684.5	opt	0.0	0
20080805	3578.89	342	1.47	3578.89	opt	0.0	28
20080905	977.46	95	0.52	977.46	opt	0.0	0
20081005	444.88	11	0.22	444.88	opt	0.0	0
20081105	40692.3	1071	1.85	40692.3	opt	0.0	33
20081205	36132.2	3298	2.71	36132.2	opt	0.0	327

Table 13: Second phase of the two-phase algorithm and CPLEX comparison for the year 2008

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20090105	31586.4	428	1.56	31586.4	opt	0.0	80
20090205	33859.8	10877	7.26	33859.8	opt	0.0	1002
20090305	46407.6	2008	2.25	46407.6	opt	0.0	65
20090405	17634.1	223	1.39	17634.1	opt	0.0	0
20090505	26569.4	897	1.83	26569.4	opt	0.0	100
20090605	4834.36	19	0.75	4834.36	opt	0.0	0
20090705	3267.37	25	1.37	3267.37	opt	0.0	0
20090805	797.0	43	0.44	797.0	opt	0.0	0
20090905	243.67	10	0.16	243.67	opt	0.0	0
20091005	0.0	1	0.02	0.0	opt	0.0	0
20091105	8370.58	262	1.41	8370.58	opt	0.0	61
20091205	14565.7	376	1.52	14565.7	opt	0.0	0

Table 14: Second phase of the two-phase algorithm and CPLEX comparison for the year 2009

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20100105	57958.7	12513	10.29	57958.7	opt	0.0	3
20100205	49927.9	94	1.52	49927.9	opt	0.0	0
20100305	35705.1	7020	3.74	35705.1	opt	0.0	967
20100405	62384.1	9963	10.74	62384.1	opt	0.0	8
20100505	44429.5	13400	8.02	44429.5	opt	0.0	548
20100605	17730.4	1128	1.77	17730.4	opt	0.0	84
20100705	6313.58	300	1.44	6313.58	opt	0.0	166
20100805	25993.5	1146	1.66	25993.5	opt	0.0	77
20100905	1304.98	89	0.89	1304.98	opt	0.0	0
20101005	34125.4	2085	2.01	34125.4	opt	0.0	11
20101105	774.65	189	0.69	774.65	opt	0.0	0
20101205	47296.7	3972	2.8	47296.7	opt	0.0	1337

Table 15: Second phase of the two-phase algorithm and CPLEX comparison for the year 2010

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20110105	36407.1	537	1.64	36407.1	opt	0.0	0
20110205	19617.9	16963	19.2	19617.9	opt	4.0	6350
20110305	23295.7	1914	2.19	23295.7	opt	0.0	90
20110405	18279.1	399	1.49	18279.1	opt	0.0	14
20110505	7135.28	650	1.6	7135.28	opt	0.0	67
20110605	19742.4	6215	3.67	19742.4	opt	0.0	307
20110705	3516.78	14	1.34	3516.78	opt	0.0	0
20110805	1089.12	28	0.59	1089.12	opt	0.0	0
20110905	10067.6	178	1.33	10067.6	opt	0.0	35
20111005	300.52	6	0.16	300.52	opt	0.0	0
20111105	36427.5	887	1.9	36427.5	opt	0.0	0
20111205	1017.47	54	0.59	1017.47	opt	0.0	0

Table 16: Second phase of the two-phase algorithm and CPLEX comparison for the year 2011

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20120105	1579.88	63	0.8	1579.88	opt	0.0	0
20120205	1825.14	58	0.66	1825.14	opt	0.0	0
20120305	4585.41	12	1.38	4585.41	opt	0.0	0
20120405	7632.36	86	1.34	7632.36	opt	0.0	0
20120505	3248.93	72	1.36	3248.93	opt	0.0	0
20120605	7439.4	377	1.47	7439.4	opt	0.0	33
20120705	1593.24	280	0.93	1593.24	opt	0.0	158
20120805	967.57	16	0.37	967.57	opt	0.0	0
20120905	19063.0	760	1.7	19063.0	opt	0.0	124
20121005	0.0	1	0.02	0.0	opt	0.0	0
20121105	13041.4	18	1.43	13041.4	opt	0.0	0
20121205	14196.4	486	1.62	14196.4	opt	0.0	120

Table 17: Second phase of the two-phase algorithm and CPLEX comparison for the year2012

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20040405	3886.13	65097	94.88	3886.13	0.88%	3590.0	408368
20040505	13223.9	177213	3600.81	13223.9	0.37%	3598.0	4056598
20040605	4887.96	54400	57.68	4887.96	0.3%	3591.0	462352
20040705	1116.43	1619	1.92	1116.43	opt	0.0	766
20040805	875.4	864	1.63	875.4	opt	0.0	342
20040905	176.98	145	0.29	176.98	opt	0.0	0
20041005	107.38	49	0.11	107.38	opt	0.0	0
20041105	1058.68	2111	2.1	1058.68	opt	0.0	246
20041205	16773.9	310548	2331.01	16773.9	0.13%	3598.0	5121037

Table 18: Second phase of the two-phase algorithm and CPLEX comparison for the year 2004 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20050105	2051.01	2004	2.08	2051.01	opt	0.0	0
20050205	1952.33	3412	2.56	1952.33	opt	0.0	1236
20050305	1961.75	133	1.49	1961.75	opt	0.0	325
20050405	1058.66	617	1.62	1058.66	opt	0.0	111
20050505	3734.6	30982	22.87	3734.6	opt	1257.0	156059
20050605	1754.9	1262	1.82	1754.9	opt	0.0	775
20050705	1203.8	2091	2.07	1203.8	opt	0.0	740
20050805	367.96	49	0.43	367.96	opt	0.0	0
20050905	95.12	49	0.11	95.12	opt	0.0	0
20051005	5382.32	25520	17.64	5382.32	0.26%	3579.0	366291
20051105	42942.1	35862	30.2	42942.1	opt	0.0	0
20051205	7733.77	17597	11.08	7733.77	0.19%	3589.0	475558

Table 19: Second phase of the two-phase algorithm and CPLEX comparison for the year 2005 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20060105	7686.79	98925	200.21	7686.79	0.78%	3581.0	504252
20060205	4742.65	8682	5.36	4742.65	opt	72.0	53290
20060305	10680.5	67710	94.52	10680.5	0.65%	3588.0	609400
20060405	5045.19	40840	33.51	5045.19	opt	1338.0	116491
20060505	1768.57	562	1.62	1768.57	opt	0.0	61
20060605	1301.88	530	1.43	1301.88	opt	0.0	47
20060705	563.37	99	0.51	563.37	opt	0.0	0
20060805	837.09	241	0.79	837.09	opt	0.0	0
20060905	0.0	1	0.03	0.0	opt	0.0	0
20061005	162.57	97	0.2	162.57	opt	0.0	0
20061105	646.93	301	0.96	646.93	opt	0.0	0
20061205	800.24	289	0.88	800.24	opt	0.0	41

Table 20: Second phase of the two-phase algorithm and CPLEX comparison for the year 2006 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20070105	1225.35	625	1.45	1225.35	opt	0.0	99
20070205	635.06	86	1.15	635.06	opt	0.0	0
20070305	852.4	289	1.27	852.4	opt	0.0	0
20070405	1094.31	1019	1.75	1094.31	opt	0.0	468
20070505	11947.6	26613	23.58	11947.6	opt	644.0	1324359
20070605	6718.4	69450	96.34	6718.4	0.64%	3588.0	558376
20070705	1043.11	2	1.12	1043.11	opt	0.0	0
20070805	821.58	97	0.59	821.58	opt	0.0	0
20070905	933.4	385	1.05	933.4	opt	0.0	69
20071005	290.66	97	0.43	290.66	opt	0.0	0
20071105	0.0	1	0.02	0.0	opt	0.0	0
20071205	563.49	49	0.5	563.49	opt	0.0	0

Table 21: Second phase of the two-phase algorithm and CPLEX comparison for the year 2007 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20080105	5836.94	1425	1.82	5836.94	opt	0.0	0
20080205	11199.1	47802	53.1	11199.1	0.49%	3581.0	697934
20080305	9773.13	607	1.5	9773.13	opt	0.0	0
20080405	2152.27	532	1.43	2152.27	opt	0.0	187
20080505	1863.69	967	1.57	1863.69	opt	0.0	145
20080605	5254.42	4607	3.15	5254.42	opt	9.0	5874
20080705	7010.51	2249	2.18	7010.51	opt	0.0	1600
20080805	2356.18	1867	1.83	2356.18	opt	0.0	620
20080905	1030.11	97	0.59	1030.11	opt	0.0	0
20081005	439.4	97	0.28	439.4	opt	0.0	0
20081105	16733.6	47607	49.57	16733.6	0.19%	3597.0	9324661
20081205	18404.1	234766	1379.83	18404.1	0.41%	3575.0	1213461

Table 22: Second phase of the two-phase algorithm and CPLEX comparison for the year 2008 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20090105	18608.6	65982	81.84	18608.6	0.13%	3590.0	793652
20090205	21859.7	302830	2169.11	21859.7	0.35%	3593.0	3153436
20090305	16731.6	168860	813.64	16731.6	0.21%	3598.0	6247636
20090405	8875.95	135092	325.61	8875.95	1.12%	3588.0	352367
20090505	9617.4	146973	478.24	9617.4	0.22%	3588.0	1021343
20090605	2181.47	2845	2.4	2181.47	opt	1.0	2069
20090705	1611.24	1008	1.6	1611.24	opt	0.0	695
20090805	540.51	119	0.52	540.51	opt	0.0	0
20090905	243.83	97	0.2	243.83	opt	0.0	0
20091005	0.0	1	0.02	0.0	opt	0.0	0
20091105	3049.89	21426	13.9	3049.89	opt	1472.0	176885
20091205	11145.3	59213	76.52	11145.3	opt	526.0	1325266

Table 23: Second phase of the two-phase algorithm and CPLEX comparison for the year 2009 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20100105	45235.5	11336	6.23	45235.5	opt	0.0	993
20100205	35352.3	174260	3600.86	35352.3	0.26%	3598.0	5662872
20100305	17660.8	280980	2244.37	17660.8	0.21%	3598.0	6502525
20100405	52130.7	263023	3600.7	52130.7	0.06%	3592.0	859733
20100505	29643.9	175084	3600.83	29644.1	0.2%	3587.0	1371664
20100605	14339.9	80099	120.43	14339.9	0.67%	3584.0	508552
20100705	4319.06	3080	2.7	4319.06	opt	1.0	2211
20100805	19259.8	275280	1601.24	19259.8	0.55%	3579.0	1101538
20100905	1126.92	337	1.06	1126.92	opt	0.0	0
20101005	16738.1	110896	272.41	16738.1	0.17%	3598.0	9117219
20101105	573.24	193	0.78	573.24	opt	0.0	0
20101205	35573.6	257868	1608.74	35573.6	opt	0.0	497

Table 24: Second phase of the two-phase algorithm and CPLEX comparison for the year 2010 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20110105	25522.0	176173	3600.85	25522.0	0.29%	3598.0	4627763
20110205	16692.6	281983	1448.87	16692.6	0.55%	3585.0	1012552
20110305	18458.5	70311	119.83	18458.5	0.29%	3587.0	1325052
20110405	13405.5	69166	124.72	13405.5	0.3%	3599.0	7889852
20110505	5111.46	10340	6.35	5111.46	opt	49.0	20192
20110605	18259.0	156836	607.93	18259.0	0.44%	3577.0	735088
20110705	2689.1	867	1.67	2689.1	opt	0.0	121
20110805	974.48	160	0.67	974.48	opt	0.0	0
20110905	7047.18	6306	4.44	7047.18	opt	11.0	20848
20111005	263.97	97	0.2	263.97	opt	0.0	0
20111105	25420.0	16190	9.29	25420.0	opt	0.0	0
20111205	874.17	97	0.69	874.17	opt	0.0	0

Table 25: Second phase of the two-phase algorithm and CPLEX comparison for the year 2011 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
20120105	1077.32	289	0.95	1077.32	opt	0.0	0
20120205	1069.02	241	0.79	1069.02	opt	0.0	0
20120305	2016.67	934	1.68	2016.67	opt	0.0	143
20120405	4376.02	3520	2.62	4376.02	opt	1.0	1407
20120505	2997.37	723	1.63	2997.37	opt	0.0	477
20120605	3858.07	10168	5.93	3858.07	opt	174.0	28800
20120705	1496.09	385	1.04	1496.09	opt	0.0	5
20120805	928.86	97	0.43	928.86	opt	0.0	0
20120905	13474.6	47333	49.65	13474.6	0.4%	3584.0	634456
20121005	0.0	1	0.04	0.0	opt	0.0	0
20121105	5490.87	4090	2.7	5490.87	opt	0.0	0
20121205	7078.94	12080	6.71	7078.94	opt	1728.0	171382

Table 26: Second phase of the two-phase algorithm and CPLEX comparison for the year 2012 with modified prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
7-0806	-	9421	2.75	-	opt	0.0	0
8-0609	4876.22	18	0.16	4876.22	opt	0.0	0
9-0206	9244.58	1519	0.53	9244.58	opt	0.0	377
9-0210	11750.4	1714	0.5	11767.6	opt	0.0	736
9-0410	12642.4	2550	0.91	12647.3	opt	0.0	1801
9-0414	9259.54	1083	0.46	9276.1	opt	0.0	159
9-0605	12600.0	1211	0.53	12600.0	opt	0.0	45
9-1013	0.0	1	0.01	0.0	opt	0.0	0
18-0605	21995.5	1	0.01	21995.5	opt	0.0	0
18-0609	192178.0	1	0.01	192178.0	opt	0.0	0
18-0811	18971.0	1	0.01	18971.0	opt	0.0	0
19-0414	250666.0	1	0.01	250666.0	opt	0.0	0
19-0609	22348.8	1	0.01	22348.8	opt	0.0	0
20-0206	95899.9	1	0.0	95899.9	opt	0.0	0
20-0210	102449.0	1	0.01	102449.0	opt	0.0	0
20-0605	125212.0	1	0.01	125212.0	opt	0.0	0
20-0806	-5402.14	1	0.01	-5402.14	opt	0.0	0
20-0811	1492.99	1	0.01	1492.99	opt	0.0	0
21-0414	24791.1	1	0.01	24791.1	opt	0.0	0
21-1013	3079.3	1	0.01	3079.3	opt	0.0	0
22-0206	70339.8	1	0.01	70339.8	opt	0.0	0
22-0210	100831.0	1	0.01	100831.0	opt	0.0	0
22-0410	29023.1	1	0.01	29023.1	opt	0.0	0
22-0605	531174.0	1	0.02	531174.0	opt	0.0	0
22-0806	1590.04	1	0.01	1590.04	opt	0.0	0
22-1010	3160.17	1	0.01	3160.17	opt	0.0	0
23-0811	30152.2	1	0.01	30152.2	opt	0.0	0
23-1010	588.49	1	0.01	588.49	opt	0.0	0
23-1013	4528.74	1	0.01	4528.74	opt	0.0	0
24-0206	269050.0	1	0.01	269050.0	opt	0.0	0
24-0414	1786550.0	1	0.01	1786550.0	opt	0.0	0
24-0609	316690.0	1	0.01	316690.0	opt	0.0	0
24-0806	52840.1	1	0.01	52840.1	opt	0.0	0
25-0210	459095.0	1	0.01	459095.0	opt	0.0	0
25-0410	792737.0	1	0.01	792737.0	opt	0.0	0

Table 27: Second phase of the two-phase algorithm and CPLEX comparison on EDF instances with default prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
26-0806	42336.3	1	0.01	42336.3	opt	0.0	0
26-1010	41206.6	1	0.01	41206.6	opt	0.0	0
26-1013	40192.6	1	0.01	40192.6	opt	0.0	0
27-0206	71323.3	1	0.01	71323.3	opt	0.0	0
27-0414	45201.6	1	0.0	45201.6	opt	0.0	0
27-0605	4628520.0	1	0.01	4628520.0	opt	0.0	0
28-0210	35630.3	1	0.02	35630.3	opt	0.0	0
28-0410	28466.4	1	0.02	28466.4	opt	0.0	0
28-0609	4646940.0	1	0.01	4646940.0	opt	0.0	0
28-0811	914930.0	1	0.01	914930.0	opt	0.0	0
29-0605	21648.4	1	0.01	21648.4	opt	0.0	0
29-1010	25714.9	1	0.01	25714.9	opt	0.0	0
29-1013	25059.9	1	0.01	25059.9	opt	0.0	0
30-0609	15452.5	1	0.01	15452.5	opt	0.0	0
31-0806	935608.0	1	0.01	935608.0	opt	0.0	0
32-0206	226313.0	1	0.01	226313.0	opt	0.0	0
32-0414	379220.0	1	0.01	379220.0	opt	0.0	0
33-0210	204167.0	1	0.01	204167.0	opt	0.0	0
33-0410	703224.0	1	0.01	703224.0	opt	0.0	0
34-0206	95178.2	1	0.01	95178.2	opt	0.0	0
34-0414	171476.0	1	0.01	171476.0	opt	0.0	0
34-0811	-2612.74	1	0.01	-2612.74	opt	0.0	0
34-1010	837867.0	1	0.02	837867.0	opt	0.0	0
34-1013	1040800.0	1	0.02	1040800.0	opt	0.0	0
35-0210	77822.2	1	0.01	77822.2	opt	0.0	0
35-0410	87091.2	1	0.01	87091.2	opt	0.0	0
35-0609	16093.3	368	0.17	16119.0	opt	0.0	0
36-1010	4976.64	1	0.01	4976.64	opt	0.0	0
36-1013	4914.43	1	0.01	4914.43	opt	0.0	0
37-0806	-3209.93	1	0.01	-3209.93	opt	0.0	0
39-0206	64146.1	4139	2.15	64146.1	opt	0.0	0
39-0414	16373.1	1	0.01	16373.1	opt	0.0	0
40-0210	84971.2	328786	3600.7	84971.2	opt	0.0	1000
40-0410	33343.4	231	0.38	33343.4	opt	0.0	0
42-1013	1766.71	1	0.01	1766.71	opt	0.0	0

Table 28: Second phase of the two-phase algorithm and CPLEX comparison on EDF instances with default prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
7-0806	-	9007	2.92	-	opt	0.0	0
8-0609	11721.7	161	0.18	11721.7	opt	0.0	0
9-0206	13830.5	265	0.21	13830.5	opt	0.0	140
9-0210	18267.6	465	0.29	18267.6	opt	0.0	998
9-0410	20367.3	793	0.35	20367.3	opt	0.0	1287
9-0414	14908.0	178	0.22	14908.0	opt	0.0	358
9-0605	27196.6	131	0.17	27196.6	opt	0.0	46
9-1013	0.0	1	0.01	0.0	opt	0.0	0
18-0605	23453.2	1	0.29	23453.2	opt	0.0	0
18-0609	192178.0	1	0.01	192178.0	opt	0.0	0
18-0811	18971.0	1	0.01	18971.0	opt	0.0	0
19-0414	250666.0	1	0.01	250666.0	opt	0.0	0
19-0609	23684.9	1	0.29	23684.9	opt	0.0	0
20-0206	95899.9	1	0.01	95899.9	opt	0.0	0
20-0210	102449.0	1	0.01	102449.0	opt	0.0	0
20-0605	125212.0	1	0.01	125212.0	opt	0.0	0
20-0806	-5402.14	1	0.01	-5402.14	opt	0.0	0
20-0811	6099.47	12	0.3	6099.47	opt	0.0	0
21-0414	26210.2	1	0.29	26210.2	opt	0.0	0
21-1013	3079.3	1	0.01	3079.3	opt	0.0	0
22-0206	70604.8	1	0.2	70604.8	opt	0.0	0
22-0210	101147.0	1	0.19	101147.0	opt	0.0	0
22-0410	32197.8	1	0.3	32197.8	opt	0.0	0
22-0605	531256.0	1	1.14	531256.0	opt	0.0	0
22-0806	5226.07	4	0.3	5226.07	opt	0.0	0
22-1010	3160.17	1	0.01	3160.17	opt	0.0	0
23-0811	32911.5	1	0.67	32911.5	opt	0.0	0
23-1010	588.49	1	0.01	588.49	opt	0.0	0
23-1013	4528.74	1	0.01	4528.74	opt	0.0	0
24-0206	269050.0	1	0.01	269050.0	opt	0.0	0
24-0414	1786550.0	1	0.01	1786550.0	opt	0.0	0
24-0609	318115.0	1	1.04	318115.0	opt	0.0	0
24-0806	52840.1	1	0.01	52840.1	opt	0.0	0
25-0210	459095.0	1	0.01	459095.0	opt	0.0	0
25-0410	792737.0	1	0.01	792737.0	opt	0.0	0

Table 29: Second phase of the two-phase algorithm and CPLEX comparison on EDF instances with lagrangian prices

	Two-phase algorithm			CPLEX			
instance	value	#iter	time	value	gap	time (s)	#nodes
26-0806	43553.0	1	0.68	43553.0	opt	0.0	0
26-1010	41206.6	1	0.01	41206.6	opt	0.0	0
26-1013	40192.6	1	0.01	40192.6	opt	0.0	0
27-0206	71559.6	1	0.46	71559.6	opt	0.0	0
27-0414	45201.6	1	0.0	45201.6	opt	0.0	0
27-0605	4632360.0	1	0.11	4632360.0	opt	0.0	0
28-0210	35630.3	1	0.02	35630.3	opt	0.0	0
28-0410	29356.2	1	1.4	29356.2	opt	0.0	0
28-0609	4656880.0	1	0.22	4656880.0	opt	0.0	0
28-0811	922252.0	1	0.67	922252.0	opt	0.0	0
29-0605	28855.2	1	0.83	28855.2	opt	0.0	0
29-1010	25943.4	1	0.73	25943.4	opt	0.0	0
29-1013	25611.8	1	0.7	25611.8	opt	0.0	0
30-0609	24340.9	1	0.81	24340.9	opt	0.0	0
31-0806	938766.0	1	0.4	938766.0	opt	0.0	0
32-0206	231193.0	1	0.91	231193.0	opt	0.0	0
32-0414	396255.0	1	0.38	396255.0	opt	0.0	0
33-0210	219105.0	1	0.87	219105.0	opt	0.0	0
33-0410	703336.0	1	0.13	703336.0	opt	0.0	0
34-0206	96963.1	1	0.76	96963.1	opt	0.0	0
34-0414	171476.0	1	0.01	171476.0	opt	0.0	0
34-0811	1807.23	7	0.15	1807.23	opt	0.0	0
34-1010	840307.0	1	1.63	840307.0	opt	0.0	0
34-1013	1048060.0	1	1.27	1048060.0	opt	0.0	0
35-0210	81240.0	1	0.77	81240.0	opt	0.0	0
35-0410	87091.2	1	0.01	87091.2	opt	0.0	0
35-0609	25960.7	1	0.15	25960.7	opt	0.0	0
36-1010	5966.28	1	0.76	5966.28	opt	0.0	0
36-1013	6116.11	1	0.76	6116.11	opt	0.0	0
37-0806	-2311.6	1	0.15	-2311.6	opt	0.0	0
39-0206	71455.2	321189	3601.21	71807.8	opt	0.0	144
39-0414	25697.9	1	0.3	25697.9	opt	0.0	0
40-0210	97332.2	298913	3601.14	97437.8	opt	6.0	7642
40-0410	45116.0	1	0.32	45116.0	opt	0.0	0
42-1013	4306.73	3	0.3	4306.73	opt	0.0	0

Table 30: Second phase of the two-phase algorithm and CPLEX comparison on EDF instances with lagrangian prices

References

[1] Wim van Ackooij, Claudia d'Ambrosio, Dimitri Thomopulos, and Renan Spencer Trindade. "Decomposition and shortest path problem formulation for solving the hydro unit commitment and scheduling in a hydro valley". In: *European Journal of Operational Research* 291.3 (2021), pp. 935–943.

[2] Alicia Arce, Takaaki Ohishi, and Sérgio Soares. "Optimal dispatch of generating units of the Itaipú hydroelectric plant". In: *IEEE Transactions on power systems* 17.1 (2002), pp. 154–158.

[3] John E Beasley and Nicos Christofides. "An algorithm for the resource constrained shortest path problem". In: *Networks* 19.4 (1989), pp. 379–394.

[4] Chun-tian Cheng, Sheng-li Liao, Zi-Tian Tang, and Ming-yan Zhao. "Comparison of particle swarm optimization and dynamic programming for large scale hydro unit load dispatch". In: *Energy Conversion and Management* 50.12 (2009), pp. 3007–3014.

[5] Daniel Duque, Leonardo Lozano, and Andrés L Medaglia. "An exact method for the biobjective shortest path problem for large-scale road networks". In: *European Journal of Operational Research* 242.3 (2015), pp. 788–797.

[6] David Eppstein. "Finding the k shortest paths". In: *SIAM Journal on computing* 28.2 (1998), pp. 652–673.

[7] Wei Fan, Xiaohong Guan, and Qiaozhu Zhai. "A new method for unit commitment with ramping constraints". In: *Electric Power Systems Research* 62.3 (2002), pp. 215–224.

[8] Markus Kruber, Axel Parmentier, and Pascal Benchimol. *Resource constrained shortest path algorithm for EDF short-term thermal production planning problem.* 2018. DOI: 10.48550/ARXIV.1809.00548. URL: https://arxiv.org/abs/1809.00548.

[9] Eugene L Lawler. "A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem". In: *Management science* 18.7 (1972), pp. 401–405.

[10] Juan I Pérez-Díaz, José R Wilhelmi, and Luis A Arévalo. "Optimal short-term operation schedule of a hydropower plant in a competitive electricity market". In: *Energy Conversion and Management* 51.12 (2010), pp. 2955–2966.

[11] Juan I Pérez-Díaz, José R Wilhelmi, and José Ángel Sánchez-Fernández. "Short-term operation scheduling of a hydropower plant in the day-ahead electricity market". In: *Electric Power Systems Research* 80.12 (2010), pp. 1535–1542.

[12] Deepak Rajan, Samer Takriti, et al. "Minimum up/down polytopes of the unit commitment problem with start-up costs". In: *IBM Res. Rep* 23628 (2005), pp. 1–14.

[13] Ted K Ralphs, Matthew J Saltzman, and Margaret M Wiecek. "An improved algorithm for solving biobjective integer programs". In: *Annals of Operations Research* 147.1 (2006), pp. 43–70.

[14] Celso C Ribeiro and Michel Minoux. "A heuristic approach to hard constrained shortest path problems". In: *Discrete Applied Mathematics* 10.2 (1985), pp. 125–137.

[15] Youcef Sahraoui, Pascale Bendotti, and Claudia d'Ambrosio. "Real-world hydro-power unit-commitment: Dealing with numerical errors and feasibility issues". In: *Energy* 184 (2019), pp. 91–104.

[16] Raouia Taktak and Claudia D'Ambrosio. "An overview on mathematical programming approaches for the deterministic unit commitment problem in hydro valleys". In: *Energy Systems* 8.1 (2017), pp. 57–79.

[17] Lara Turner. "Variants of the shortest path problem". In: *Algorithmic Operations Research* 6.2 (2011), pp. 91–104.

[18] Marc Visée, Jacques Teghem, Marc Pirlot, and L. Ekunda Ulungu. "Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem". In: *Journal of Global Optimization* 12.2 (1998), pp. 139–155.

[19] Gang Yu and Jian Yang. "On the Robust Shortest Path Problem". In: *Computers & Operations Research* 25.6 (1998), pp. 457–468. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(97)00085-3. URL: https://www.sciencedirect.com/science/article/pii/S0305054897000853.

[20] Xiaoyan Zhu and Wilbert E Wilhelm. "A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation". In: *Computers & Operations Research* 39.2 (2012), pp. 164–178.

[21] Xiaoyan Zhu and Wilbert E Wilhelm. "Three-stage approaches for optimizing some variations of the resource constrained shortest-path sub-problem in a column generation context". In: *European journal of operational research* 183.2 (2007), pp. 564–577.