Efficient exact A* algorithm for the single unit hydro unit commitment problem

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

Efficient exact A* algorithm for the single plant Hydro Unit Commitment problem

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

February 26, 2024

Abstract

The Hydro Unit Commitment problem (HUC) specific to hydroelectric plants is part of the electricity production planning problem, called Unit Commitment Problem (UCP). More specifically, the studied case is that of the HUC with a single plant, denoted 1-HUC. The plant is located between two reservoirs. The horizon is discretized in time periods. The plant 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 window resource constraints, defined by a minimum and maximum volume per time period. 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 exact A* variant, so called HA*, is proposed to solve the 1-HUC accounting for window constraints, with a reduced search space and a dedicated optimistic heuristic. This variant is compared to a classical Resource Constrained Shortest Path Problem (RCSPP) algorithm and a Mixed Integer Linear Programming formulation solved with CPLEX. Results show that the proposed algorithm outperforms both concurrent alternatives in terms of computational time in average on a set of realistic instances, meaning that HA* exhibits a more stable behavior with a larger number of instances solved.

1 Introduction

An electricity producer aims at meeting the demand at any time. This is because electricity can hardly be stored, meaning that any excess is lost. Scheduling the short-term production in order to meet the demand defines the Unit Commitment Problem (UCP), which is solved for the day ahead. At Electricité de France (EDF), large-scale UCP instances are solved by Lagrangian decomposition [1], yielding subproblems of the same nature (thermal, hydraulic, solar,...). As each subproblem has different constraints, specific approaches are developed for each of them. Among these subproblems, the Hydro Unit Commitment (HUC) has received a lot of attention, due to the large size of its instances. Indeed, instances of the HUC involve valleys, which can be constituted of up to twenty plants, linked with reservoirs.

At EDF, the HUC is modeled as a Mixed Integer Linear Program (MILP) and solved with CPLEX [2]. This MILP considers operating points, which are pairs (water flow, corresponding generated power). In practice, the HUC is not solved to optimality within the time limit set, as such a representation induces an exponential number of solutions and large computational times. More recent work [3] have pointed out the interest of solving the HUC using a Lagrangian relaxation algorithm, where subproblems are single plant HUC (1-HUC). The main benefit of this relaxation is that the 1-HUC can be solved with dynamic programming, while the master problem handles the coupling constraints. It is shown that such a relaxation can lead to overall better results than solving the HUC as an MILP, which emphasizes the relevance of an efficient algorithm to solve the 1-HUC.

In this paper, we consider the 1-HUC with a single plant located between two reservoirs. A diagram, taken from [4] and shown in Figure 1, is sketching the 1-HUC. The principle of hydroelectric production is the following: the water from the upstream reservoir flows into the downstream reservoir through the units of the plant, thus driving the turbines of the units, which in turn power the generator to produce electricity. When operating in reverse, the pumps of the units can move the water from the downstream reservoir to the upstream reservoir, which consumes electricity. The plant operates on M turbining points, N pumping points and an idle operating point. With $I = \{-N, \dots, 0, \dots, M\}$, each operating point $i \in I$ is defined as a pair formed by a water flow D_i and a generated power P_i. Both D_i and P_i are positive (resp. negative) for turbining (resp. pumping) operating points, i.e., with $i > 0$ (resp. $i < 0$), and are 0 for the idle operating point $i = 0$. The operating points are defined in a cumulative fashion meaning that if a plant is at turbining (resp. pumping) operating point i, then order constraints apply, involving all points $1 \leq j < i$ (resp. $-1 \geq j > i$) to also be operated. At each time period, the plant cannot turbine and pump simultaneously. The time horizon is discretized into T time periods. At each time period t, the plant turbines (resp. pumps) a water flow and produces (consumes) an amount of energy that is considered to be constant for the duration of the time period. Resource window constraints state that the volume of each reservoir $n \in \{1, 2\}$ lies between a lower bound $\underline{V}_t^n$ and an

upper bound $\overline{V}_t^n$ that are time-dependent. At each time period, the reservoir n receives an additional intake of water A_t^n. The additional intake can be positive to represent rain, melting snow etc., or negative to represent the use of water for local agriculture etc.

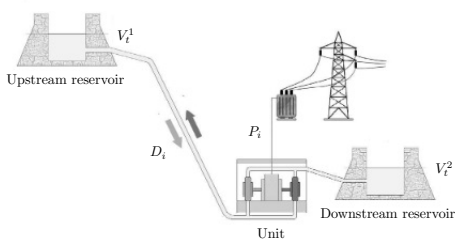


Figure 1: Diagram of the 1-HUC

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 value of the energy produced or consumed at a time-dependent unit value Λ_t. The problem is to maximize the total revenue, while satisfying the reservoir capacities at each time period.

In practice, water management policies require that reservoirs should meet target volumes ($\underline{V}_t^n = \overline{V}_t^n - \epsilon$) at the end of the time horizon. Due to these target volumes and the finite set of operating points, the 1-HUC may not have any feasible solution. However, it is possible to adjust [5] or relax [6] the target volumes in order to obtain feasible instances. Hence, we consider in this paper only 1-HUC instances admitting feasible solutions.

In this paper the aim is to propose a dynamic programming algorithm dedicated to the 1-HUC, modeled as a Resource Constrained Shortest Path Problem (RCSPP) with resource window constraints. The algorithm we propose is an exact variant of the A* algorithm [7] used to compute the shortest path in a graph. To obtain an efficient algorithm, the bounds of the windows are first tightened by propagating them from any time period to another, and a dedicated heuristic is developed. As we propose an exact algorithm, we compare it with two other exact methods, namely an MILP solved by default CPLEX as currently done at EDF and a classical RCSPP algorithm. The corresponding evaluation of performance is done on various realistic instances. The numerical results show that our algorithm yields smaller computational time variations compared to both the MILP formulation solved by CPLEX and the RCSPP algorithm.

The remainder of the paper is organized as follows. In Section 2, a literature review of dynamic programming algorithms for related problems is reported. In Section 3, a formulation of the 1-HUC is proposed. In Section 4, graph representations of the 1-HUC are provided. In Section 5, the bound tightening procedure and the exact A* variant are described. In Section 6, numerical results are presented. In Section 7, concluding remarks and perspectives for further research are drawn.

2 State of the art

In this section, we first present a literature review of dynamic programming algorithms developed for the UCP and for the HUC. As the HUC can be represented as an RCSPP, we also include in this review dynamic programming algorithms for the RCSPP.

2.1 Dynamic programming for the Unit Commitment Problem

A dynamic programming algorithm for a single Unit Commitment (1-UC) with ramp and min up/down constraints is presented in [8]. The algorithm is based on a graph with a source vertex and several groups of T vertices. For each even (resp. odd) group, vertex t indicates that the unit is turned off (resp. on) 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. The difference between the 1-UC and the 1-HUC is that there is no resource in the 1-UC, while in the 1-HUC, the presence of reservoirs with minimum and maximum volumes requires to account for the water resource.

2.2 Dynamic programming for the Hydro Unit Commitment

In this part we focus on dynamic programming algorithms for the HUC, most of them being cited in the survey [9].

In [1] the author presents a two phase approach to solve the HUC, solving an LP for the first phase and using a dynamic programming algorithm for the second phase. More precisely, the second phase consists of solving the 1-HUC for each plant of the valley with dynamic programming, aiming to get the closest solution to the LP solution while taking into account constraints omitted in the LP. For this phase, the considered underlying graph is as follows. The reservoir volume is discretized, yielding hundreds of possible volume values for a reservoir. Then, a vertex is defined for each volume value and each time period, thus leading to hundreds of vertices per time period. A Bellman-Ford algorithm [10] is used to find a path in this graph. Such a discretization discards a lot of realistic states. In this paper, we consider all possible states with respect to the operating points, which can be exponential for instances with a large number of time periods. With an exponential number of states, the Bellman-Ford algorithm becomes far less efficient.

In [6] a method for solving a non-linear 1-HUC with a target volume is described. To solve this problem with dynamic programming, a state diagram is constructed. In a similar fashion as in [1], evenly discretized volumes are considered, yielding a limited number of states per time periods. In order to have feasible solutions, the target volume is relaxed to match this discretization. The state diagram is constructed by generating the possibilities to reach the

target volume from the initial volume, satisfying 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. As we consider 1-HUC instances with and without target volumes, a backward algorithm may not be practical with large volumes and no target volume.

In [3], a decomposition method for solving the HUC with shortest paths is described. The considered HUC is a valley where each plant has a finite number of operating points. The topology of the valley is not restricted to a chain, as each plant (resp. reservoir) can have a set of upstream and downstream reservoirs (resp. plants). There are additional ramping constraints, namely 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 solution approach decomposes this HUC into multiple 1-HUC. Some 1-HUC without resource constraints are solved by a shortest path algorithm, while others with resource constraints are solved by a labeling algorithm defined in [11]. The latter algorithm is adapted from a classical RCSPP algorithm [12] to take into account a minimum bound for the resource. It is mentioned that this labeling algorithm loses its dominance properties between two labels if one of them does not verify the minimum bound on the resource. Such a case is more frequent when the target volume is considered with equality constraints, making this algorithm less efficient.

There are also other problems solved by dynamic programming, related to the 1-HUC. In [13] a dynamic programming approach is described to solve an HUC on instances of the Itaipù plant (Brazil, Paraguay). This problem differs from ours as the only constraint is to satisfy the minimum and maximum number of turbines running at each time period. No volume is considered, therefore there are neither bounds on the volume, nor a target volume. In [14] the Hydro Unit Load Dispatch problem (HULD) is presented. This problem differs from the HUC, as the water flow is known, and solving the HULD is to provide the most economic distribution of the water through the different turbines, while verifying the flow capacity of the turbines at each time period.

2.3 Shortest path with resource constraints

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

There are works on the RCSPP to solve the thermal problem on EDF instances [15]. In that paper it is indicated that the resource has an upper bound but no lower bound. However, as specified in [3], the difficulty of the HUC comes from the lower bound on the volume, which prevents the use of dominance rules.

In the survey [16], a state of the art review of 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 (such as in the HUC).

Three papers are cited, namely [17] describing a heuristic, [18] presenting an integer formulation and [19] proposing a dynamic programming algorithm. As we look for an exact algorithm, we will focus on the three-phase algorithm described in [19]. The presented algorithm solves the RCSPP with window constraints on acyclic graphs. The main idea, further detailed in [20], is to extend the graph, such that if multiple paths lead to the same vertex from the source, a new vertex is created for each of these paths. Once the graph has been extended in this way, the problem is solved with a pseudo-polynomial time algorithm. Such a graph extension seems difficult to apply to the 1-HUC. Indeed, graph representations of the 1-HUC (see Section 4) solely have arcs from time period t to time period $t+1$. Hence, the extension can lead up to M^T vertices at time period T. Even a pseudo-polynomial algorithm on the extended graph could be impractical in the case of the 1-HUC.

3 Integer linear programming

With $x_{t,i}$ the binary variable indicating whether the plant is at least at operating point $i \in I$ at time period $t \leq T$, we obtain the following formulation:

$$\begin{aligned}
\max\ & \sum_{t=1}^{T}\sum_{i\in I}\Lambda_t P_i x_{t,i} + \Phi^1\Big(\sum_{t=1}^{T}(A_t^1 - \sum_{i\in I} D_i x_{t,i})\Big) \\
& + \Phi^2\Big(\sum_{t=1}^{T}(A_t^2 + \sum_{i\in I} D_i x_{t,i})\Big) \\
\text{s.c.}\quad & V_0^1 + \sum_{t=1}^{t'}(A_t^1 - \sum_{i\in I} D_i x_{t,i}) \leq \overline{V}_{t'}^1, \quad \forall t' \leq T && \text{(a1)} \\
& V_0^1 + \sum_{t=1}^{t'}(A_t^1 - \sum_{i\in I} D_i x_{t,i}) \geq \underline{V}_{t'}^1, \quad \forall t' \leq T && \text{(b1)} \\
& V_0^2 + \sum_{t=1}^{t'}(A_t^2 + \sum_{i\in I} D_i x_{t,i}) \leq \overline{V}_{t'}^2, \quad \forall t' \leq T && \text{(a2)} \\
& V_0^2 + \sum_{t=1}^{t'}(A_t^2 + \sum_{i\in I} D_i x_{t,i}) \geq \underline{V}_{t'}^2, \quad \forall t' \leq T && \text{(b2)} \\
& x_{t,i} \geq x_{t,i+1}, \quad \forall t \leq T, \forall i \in \{1,\dots,M-1\} && \text{(c)} \\
& x_{t,i} \geq x_{t,i-1}, \quad \forall t \leq T, \forall i \in \{-1,\dots,-N+1\} && \text{(d)} \\
& x_{t,1} + x_{t,-1} \leq 1, \quad \forall t \leq T && \text{(e)} \\
& x_{t,i} \in \{0,1\}, \quad \forall t \leq T, \forall i \in I && \text{(f)}
\end{aligned}$$

In this formulation, the objective function maximizes the total revenue. Constraints (a1) to (b2) ensure that the minimum/maximum bounds on the volume

are verified for both the upstream and downstream reservoirs at each time period. Constraints (c) and (d) correspond to the order of the operating points. Constraints (e) prevent the plant from turbining and pumping simultaneously. Lastly, constraints (f) indicate that all variables $x_{t,i}$ are binary.

3.1 Shifting all operating points

In the following, it will be convenient to only have operating points with non-negative power and flow. In [3] a modification on the flows and the volume bounds is considered to only have such operating points. First, each turbining operating point i, for $i \in \{1, .., M\}$ is renumbered $i + N$, yielding operating point $(D'_{i+N},\ P'_{i+N})$ with $D'_{i+N} = D_i$ and $P'_{i+N} = P_N$. For each pumping operating point $i \in \{-N, \ldots, -1\}$, a turbining operating point numbered $N + i+1$ is created, yielding operating point $(D'_{N+i+1},\ P'_{N+i+1})$ with $D'_{N+i+1} = |D_i|$ and $P'_{N+i+1} = |P_N|$. Operating point 0 remains unchanged. As such, all the operating points have non-negative cumulated flows. For a given $t \leq T$, the bounds on the volume $\overline{V}^1_t$ and $\underline{V}^1_t$ (resp. $\overline{V}^2_t$ and $\underline{V}^2_t$) are shifted in order to keep the same feasible solutions: $\overline{V'}^1_t = \overline{V}^1_t - t\sum_{i=-1}^{-N} |D_i|$ and $\underline{V'}^1_t = \underline{V}^1_t - t\sum_{i=-1}^{-N} |D_i|$ (resp. $\overline{V'}^2_t = \overline{V}^2_t + t\sum_{i=-1}^{-N} |D_i|$ and $\underline{V'}^2_t = \underline{V}^2_t + t\sum_{i=-1}^{-N} |D_i|$).

Example 1. Consider an instance of the 1-HUC with $M = 3$ turbining operating points $(D_1 = 3, P_1 = 3)$, $(D_2 = 4, P_2 = 3)$, $(D_3 = 2, P_3 = 2)$ and $N = 2$ pumping operating points $(D_{-1} = -3, P_{-1} = -5)$, $(D_{-2} = -2, P_{-2} = -4)$. The initial volumes are $V^1_0 = 50$, $V^2_0 = 30$. With $T = 3$, upstream reservoir bounds are $\overline{V}^1 = [100, 100, 20]$ and $\underline{V}^1 = [0, 0, 20]$, and downstream reservoir bounds are $\overline{V}^2 = [60, 60, 60]$ and $\underline{V}^2 = [0, 0, 0]$.

The renumbering is such that the 3 turbining operating points, of index 1 to 3 are now of index $1 + N$ to $3 + N$, i.e., $(D'_3 = 3, P'_3 = 3)$, $(D'_4 = 4, P'_4 = 3)$ and $(D'_5 = 2, P'_5 = 2)$ From pumping operating point -1 a turbining operating point $N - 1 + 1 = 2$ is created, and similarly from the pumping operating point -2 operating point $N - 2 + 1 = 1$ is created, i.e., $(D'_1 = 2, P'_1 = 4)$ and $(D'_2 = 3, P'_2 = 5)$.

The upstream reservoir upper bounds are modified as follows $\overline{V'}^1_1 = \overline{V}^1_1 - 1 \cdot (2+3) = 95$, $\overline{V'}^1_2 = \overline{V}^1_2 - 2 \cdot (2+3) = 90$, $\overline{V'}^1_3 = \overline{V}^1_3 - 3 \cdot (2+3) = 5$. Similarly, we obtain the following upstream reservoir lower bounds $\underline{V'}^1 = [-5, -10, 5]$. The shift is done in the opposite way for the downstream reservoir, $\overline{V'}^2 = [65, 70, 75]$ and $\underline{V'}^2 = [5, 10, 15]$.

In the following, we only refer to the 1-HUC with operating points of non-negative flow and power, i.e., with $I = \{0, \ldots, M\}$. In this case, only constraints (a1) to (b2), (c) and (f) are necessary to model the constraints of the 1-HUC, and no renumbering is required.

3.2 Rewriting the formulation

The formulation defined previously is a classical MILP model for the 1-HUC. We rewrite the formulation in order for the window constraints to appear more clearly. Constraints (a1), (a2), (b1) and (b2) can be rewritten as follows, considering $\underline{V'}^1_t$ $\overline{V'}^1_t$, $\underline{V'}^2_t$ and $\overline{V'}^2_t$:

$$\sum_{t=1}^{t'}\sum_{i=1}^{M} D_i x_{t,i} \geq V_0^1 + \sum_{t=1}^{t'} A_t^1 - \overline{V'}^1_{t'}, \quad \forall t' \leq T \tag{a1'}$$

$$\sum_{t=1}^{t'}\sum_{i=1}^{M} D_i x_{t,i} \leq V_0^1 + \sum_{t=1}^{t'} A_t^1 - \underline{V'}^1_{t'}, \quad \forall t' \leq T \tag{b1'}$$

$$\sum_{t=1}^{t'}\sum_{i=1}^{M} D_i x_{t,i} \leq \overline{V'}^2_{t'} - V_0^2 - \sum_{t=1}^{t'} A_t^2, \quad \forall t' \leq T \tag{a2'}$$

$$\sum_{t=1}^{t'}\sum_{i=1}^{M} D_i x_{t,i} \geq \underline{V'}^2_{t'} - V_0^2 - \sum_{t=1}^{t'} A_t^2, \quad \forall t' \leq T \tag{b2'}$$

There are redundancies between (a1') and (b2'), and between (a2') and (b1'). Let us introduce bounds $\beta_{t'}$ and $\alpha_{t'}$ in the following way with $t' \leq T$:

$$\beta_{t'} = \max(V_0^1 + \sum_{t=1}^{t'} A_t^1 - \overline{V'}^1_{t'}, \underline{V'}^2_{t'} - V_0^2 - \sum_{t=1}^{t'} A_t^2)$$

$$\alpha_{t'} = \min(V_0^1 + \sum_{t=1}^{t'} A_t^1 - \underline{V'}^1_{t'}, \overline{V'}^2_{t'} - V_0^2 - \sum_{t=1}^{t'} A_t^2)$$

By using $\beta_{t'}$ and $\alpha_{t'}$, and rewriting the objective function, we obtain the formulation F_{1HUC} defined as follows:

$$\begin{aligned}
\max \ & \sum_{t=1}^{T}\sum_{i=1}^{M} (\Lambda_t P_i - \Phi^1 D_i + \Phi^2 D_i) x_{t,i} \\
& + \Phi^1 \sum_{t=1}^{T} A_t^1 + \Phi^2 \sum_{t=1}^{T} A_t^2 \\
\text{s.c.} \quad & \sum_{t=1}^{t'}\sum_{i=1}^{M} D_i x_{t,i} \geq \beta_{t'} \quad \forall t' \leq T && \text{(a)} \\
& \sum_{t=1}^{t'}\sum_{i=1}^{M} D_i x_{t,i} \leq \alpha_{t'} \quad \forall t' \leq T && \text{(b)} \\
& x_{t,i} \geq x_{t,i+1} \quad \forall t \leq T, \forall i \leq M-1 && \text{(c)} \\
& x_{t,i} \in \{0,1\} \quad \forall t \leq T, \forall i \leq M && \text{(f)}
\end{aligned}$$

With this formulation, the objective function is to maximize the value of each active operating point, plus a constant. Also, one can see (a) and (b) as resource window constraints, or (a) as nested cover constraints and (b) as nested knapsack constraints. This formulation can also be improved by tightening the window constraints, as shown at the end of Section 5.1.

4 Graph representation

The 1-HUC can be represented graphically in two ways, namely in a fashion similar either to the knapsack problem, or to the RCSPP. In the following we describe both representations and their dominance rules. We first introduce the cumulated flows between two time periods.

Definition 1 (Cumulated flow $\mathcal{D}_{t',t}$)**.** The cumulated flow $\mathcal{D}_{t',t}$ is the sum of the flows from time periods t' to t:

$$\mathcal{D}_{t',t} = \sum_{t''=t'}^{t} \sum_{i=1}^{M} D_i x_{t'',i}$$

4.1 Representation as a knapsack problem

Let $G_{KP} = (V_{KP}, A_{KP})$ defined as follows. Each vertex $u \in V_{KP}$ is defined as a pair $u = (t, d)$ with t the time period, and d the cumulated flow $\mathcal{D}_{1,t}$ with variables x associated to a path that reaches u. Without loss of generality, $u = (t, d)$ is considered only if $d \in [\beta_t; \alpha_t]$. The source vertex s is defined as $s = (0, 0)$. For each vertex $u = (t, d)$ and $v = (t{+}1, d{+}\sum_{j=0}^{i} D_j)$ with $t < T$ and $i \in \{0, \ldots, M\}$ there is an arc $(u, v) \in A_{KP}$, of value $\sum_{j=0}^{i} \Lambda_t P_j - \Phi^1 D_j + \Phi^2 D_j$.

The downside of such a graph is its exponential number of vertices. However, by tightening the bounds as shown in Section 5.1, it is possible to drastically reduce the number of vertices in G_{KP}. Furthermore, we can use classical dominance rules for the longest path:

Definition 2 (Dominance rule 1)**.** Let p and q be two paths from s to a vertex u. By induction the path with the lowest value is dominated, as it cannot lead to an optimal solution.

Definition 3 (Dominance rule 2)**.** Let p be a path from s to u going through a vertex v, and q be a path from s to v. Let $p_{s,v}$ be the subpath of p, from s to v and $p_{v,u}$ the subpath of p from v to u. If the value of $p_{s,v}$ is larger than that of q, then q is dominated. If the value of q is larger than that of $p_{s,v}$, then p is dominated by the path concatenating q and $p_{v,u}$.

Example 2. Consider an instance of the 1-HUC with $T = 3$, $M = 3$. The operating points are $(4, 4)$, $(6, 5)$ and $(8, 6)$. The bounds are $\beta_{=}[0, 0, 5]$ and $\alpha_{=}[12, 12, 10]$. Fig. 2a represents the graph G_{KP} associated to this instance.

4.2 Representation as an RCSPP

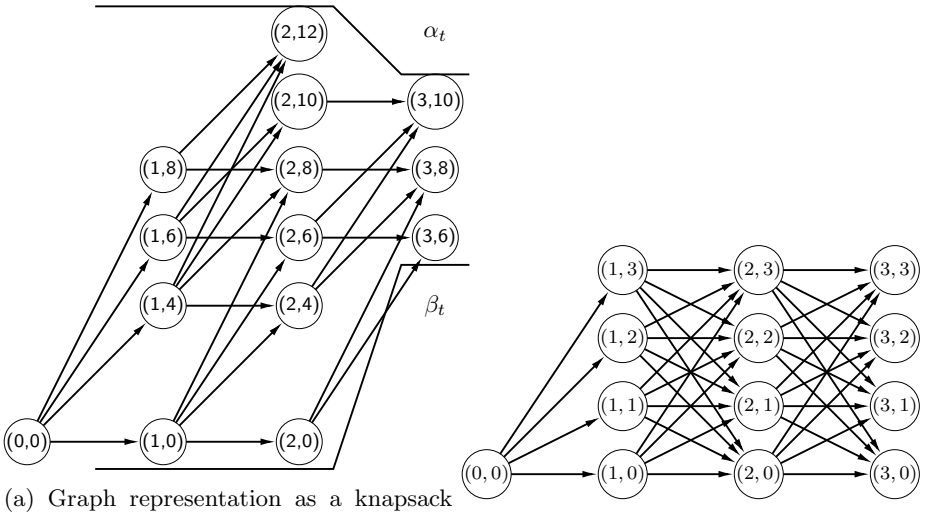
Let $G_R = (V_R, A_R)$ be defined as follows. Each vertex $u \in V_R$ is defined as a pair $u = (t, i)$, with t the time period and i the operating point. The source s is defined as $s = (0, 0)$. From each vertex $u = (t, i)$, with $t < T$, and $v = (t+1, i')$ with $i' \in \{0, \ldots, M\}$ there is an arc $(u, v) \in A_R$. Arc a towards v has value $\sum_{j=0}^{i'} \Lambda_t P_j - \Phi^1 D_j + \Phi^2 D_j$ and consumes $\sum_{j=0}^{i'} D_j$ amount of the resource.

The downside of G_R, is that there are paths in this graph which do not verify the resource constraints, hence one needs to verify the resource constraints for each path. Note that the resource in this case is $\mathcal{D}_{1,t}$ for variables x associated to the path. On the positive side, the number of vertices is polynomial, and it is possible to use a classical dominance rule for the RCSPP as defined in [3]:

Definition 4 (Dominance rule 3)**.** If there are two partial paths from s to the same vertex u, by induction if a path has a lower value and uses more resource than another one, then this path is dominated provided both partial paths use sufficient resource to verify all lower bounds on the resource.

Note that the condition on the resource usage to ensure that the lower bounding constraints (a) are satisfied seriously weakens the dominance rules when these constraints are active.

Example 3. Consider the instance of Example 2. Fig. 2b represents the graph G_R associated to this instance.



(a) Graph representation as a knapsack problem

(b) Graph representation as an RCSPP

Figure 2: Graph representations of the 1-HUC

There are two other graph representations for the 1-HUC described in the literature. In [20], the original graph defined for the RCSPP is similar to the

one depicted in Fig. 2b. However, as mentioned in Section 2.3, this graph is extended such that if multiple paths exist from vertex s to a vertex u, then u is duplicated such that only a single path leads to each vertex. The extended graph would be larger than the one depicted in Fig. 2a, as even with an exponential number of vertices, there are still multiple paths between s and most of the vertices. In [6], as described in Section 2.2, the volume is evenly discretized. The resulting graph has similar vertices as in Fig. 2a, but at each time period, vertices only exist for volumes within the set of discretized volumes. In the experimental results of [6], it is stated that the discretization is ranging from 0.3% to 0.5% of the difference between the minimum and maximum volume of the reservoir, which represents about 300 vertices at each time period. Such a graph heavily differs from the graph we consider, as there can be a very large number of vertices at each time period.

5 Exact A* variant for the 1-HUC

In this section, we describe the new algorithm proposed to solve the 1-HUC. The aim of this algorithm is to find the longest path in graph G_{KP} as described in Section 4.1. A difficulty of this graph is the exponential number of vertices, which is why we resort to a variant of the A* algorithm [7]. The A* algorithm is particularly efficient when the number of vertices is large as it involves an optimistic heuristic to guide the search, and to discard sub-optimal partial solutions. We denote the proposed exact variant of the A* algorithm for the 1-HUC by HA*. This algorithm involves a dedicated optimistic heuristic for the 1-HUC. To further improve the performance of this algorithm, we also present a pre-processing bound tightening procedure in order to reduce the number of vertices in G_{KP}.

In the following, we first present the bound tightening procedure, then the optimistic heuristic, i.e., an upper bound on the optimal value and finally the HA* algorithm.

5.1 Tightening the bounds

The bounds α_t and β_t from inequalities (a) and (b) can directly be used to prune infeasible vertices. Thus, tightening these bounds may lead to a smaller number of vertices, thus reducing the search space.

In order to tighten the bounds, we use the cumulated flow $\mathcal{D}_{t',t}$ as previously defined. At each time period, the flow is between 0 and $\sum_{i=1}^{M} D_i$. For any pair of time periods (t', t), $t' < t$, the lower bound $\underline{\mathcal{D}}_{t',t} = 0$ and the upper bound $\overline{\mathcal{D}}_{t',t} = (t - t' + 1) \sum_{i=1}^{M} D_i$ will never be violated by a feasible solution. Note that α_t and β_t are not bounds on the flow at time period t, but rather bounds on $\mathcal{D}_{1,t}$. If the gap between β_t and α_t is large, then one may develop a large number of vertices, sometimes leading to intractable instances. For example, in the case of the HUC, it is very common to have upper bounds α_t very large compared to the flows, and negative lower bounds β_t. We can therefore introduce bounds

$\hat{\alpha}_t$ and $\hat{\beta}_t$ in the following way:

$$\hat{\alpha}_t = \min(\alpha_t, \overline{\mathcal{D}}_{1,t})$$
$$\hat{\beta}_t = \max(\beta_t, \underline{\mathcal{D}}_{1,t})$$

By using bounds $\hat{\alpha}_t$ and $\hat{\beta}_t$, we can drastically reduce the number of vertices. However, it is still possible to further reduce it. Suppose that at time period t the bounds are such that $\hat{\beta}_t > \hat{\beta}_{t+1}$. As the water flows are all non-negative, any vertex $u = (t+1, d)$ with $d < \hat{\beta}_t + \underline{\mathcal{D}}_{t+1,t+1}$ cannot be part of a feasible solution. Similarly, if $\hat{\beta}_{t-1} < \hat{\beta}_t$, any vertex $u = (t-1, d)$ with $d < \hat{\beta}_t - \overline{\mathcal{D}}_{t,t}$ cannot be part of a feasible solution. Extending this logic to the upper bounds on d, we can tighten the bounds of any time period from the bounds of any other time period, following the rules below. Let a pair of time periods (t', t) with $t' < t$. Then, $\mathcal{D}_{1,t}$ must stay in $[\hat{\beta}_{t'} + \underline{\mathcal{D}}_{t'+1,t}; \hat{\alpha}_{t'} + \overline{\mathcal{D}}_{t'+1,t}]$ and $\mathcal{D}_{1,t'}$ lies in $[\hat{\beta}_t - \overline{\mathcal{D}}_{t'+1,t}; \hat{\alpha}_t - \underline{\mathcal{D}}_{t'+1,t}]$.

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

$$\tilde{\alpha}_t = \min(\min_{t'<t}(\hat{\alpha}_{t'} + \overline{\mathcal{D}}_{t'+1,t}), \min_{t'>t}(\hat{\alpha}_{t'} - \underline{\mathcal{D}}_{t+1,t'}))$$
$$\tilde{\beta}_t = \max(\max_{t'<t}(\hat{\beta}_{t'} + \underline{\mathcal{D}}_{t'+1,t}), \max_{t'>t}(\hat{\beta}_{t'} - \overline{\mathcal{D}}_{t+1,t'})$$

Tight bounds α_t^* and β_t^* are calculated as follows:

$$\alpha_t^* = \min(\hat{\alpha}_t, \tilde{\alpha}_t)$$
$$\beta_t^* = \max(\hat{\beta}_t, \tilde{\beta}_t)$$

Note that computing all bounds β_t^* is of complexity T^2. Indeed, for a given t, computing β_t as well as $\tilde{\beta}_t$ both require one comparison, computing $\tilde{\beta}_t$ needs T comparisons and computing β_t^* requires one comparison. Hence, computing all β_t^* is of complexity T^2. We obtain a similar complexity for upper bounds α_t^*.

Example 4. Let us define an instance of the 1-HUC with $T = 6$. Bounds are $\beta_3 = 2$, $\beta_6 = 5$, $\alpha_3 = 2$, $\alpha_6 = 5$, and $\beta_t = -2$, $\alpha_t = 10$ for t in $\{1, 2, 4, 5\}$. The operating points are such that at each time period, the maximum flow is 2. By applying tighter bounds we can see that we drastically reduce the possibilities, thus the number of vertices potentially developed by dynamic programming. In Table 1a the invalid values for the total flow at each time period, with respect to bounds β_t, are marked with a cross. Table 1b is similar to Table 1a with tighter bounds $\hat{\alpha}_t$ and $\hat{\beta}_t$, the crosses being in bold to emphasize the tightening of the bounds. Table 1c follows the same representation with the tightest bounds β_t^* and α_t^*.

As previously mentioned, formulation F_{1HUC} can be improved, considering bounds β_t^* and α_t^* instead of β_t and α_t in constraints (a) and (b). We denote by F_{1HUC}^+ the formulation F_{1HUC} with the bound tightening.

In the following, we consider the graph G_{KP} with the tightest bounds β_t^* and α_t^*, denoted by G_{KP}^*.

Table 1: Reducing the search space using bounds on the flows

(a) Table with bounds α_t and β_t

t	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
0															
1	X														X
2	X														X
3	X	X	X	X	X		X	X	X	X	X	X	X	X	X
4	X														X
5	X														X
6	X	X	X	X	X	X	X	X		X	X	X	X	X	X

(b) Table with bounds $\hat{\alpha}_t$ and $\hat{\beta}_t$

t	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
0															
1	X	**X**	**X**				**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	X
2	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
5	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^*

t	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
0															
1	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

5.2 Optimistic heuristic

In the case of the 1-HUC, an optimistic heuristic overestimates the value of the objective function because we are solving a maximization problem. Let p be a path from time period 1 to t representing already taken decisions. The heuristic aims at computing an optimistic cost from time period $t+1$ to T. The idea of the proposed optimistic heuristic is to compute an improved linear relaxation on time periods $t+1$ to T. To do so, we define quadruplets $(t, i, val, flow)$, with t a time period, i an operating point, val the value of any arc $\big((t-1, d), (t, d + \sum_{j=1}^{i} D_j)\big)$ in G^*_{KP}, and $flow$ the value $\sum_{j=1}^{i} D_j$. The aim is to progressively increase the values of variables $x_{t,i}$ depending on their profitability, being $val/flow$.

Algorithm 1 describes how to compute a linear relaxation on time periods $t+1$ to T from a partial path in graph G^*_{KP}, as detailed in the following four steps.

Step 1: Initialize a fractional solution with $x_{t',i} = 1$ if the partial solution represented by path p requires operating point $i \leq M$ at time period $t' \leq t$, $x_{t',i} = 0$ otherwise. This step is represented by lines 1 to 8 of Algorithm 1

Step 2: Initialize a list with all the quadruplets at time period $t' \in [t+1; T]$ by decreasing profitability $val/flow$. This step is represented by lines 9 to 14 of Algorithm 1

Feasibility step 3: This step is repeated as long as lower bounding constraints (a) are not verified (the while loop at line 15 of Algorithm 1). The algorithm looks for the smallest time period t' such that (a) is not satisfied, and for $x_{t'',i}$ with $t'' \in [t+1; t']$ maximizing profitability $val/flow$. The variable $x_{t'',i}$ is increased depending on its profitability:

- If the profitability is positive, fractionally increase $x_{t'',i}$ as much as possible provided all upper bounds α are satisfied.
- Otherwise, fractionally increase $x_{t'',i}$ as little as possible provided all upper bounds and the lower bound $\beta_{t'}$ are satisfied.

All variables $x_{t'',i'} < x_{t'',i}$ with $i' < i$ must be set to the same value as $x_{t'',i}$ in order to satisfy the order constraints. Also, for any $i' > i$, quadruplets $(t'',i',val',flow')$ are updated as $(t'',i',val'-val,flow'-flow)$. This is because as $x_{t'',i} > x_{t'',i'}$, one can increase $x_{t'',i'}$ without increasing $x_{t'',i}$ while still verifying order constraints. All variables that cannot be further increased due to the upper bounds are removed from the list.

Optimality step 4: This step is repeated as long as there is a variable of positive profitability in the list of variables (again the while loop at line 15 of Algorithm 1). Select the first variable of the list, and fractionally increase its value as much as possible provided all upper bounds are satisfied. Remove from the list all variables that cannot be further increased due to the upper bound.

Property 1. *The fractional solution returned by Algorithm 1 verifies order constraints.*

Proof. Consider two quadruplets (t,i,val,$flow$) and (t,i',val',$flow'$), with t a time period considered in the heuristic, and $i' > i$. At the start of the algorithm, $x_{t,i} = x_{t,i'} = 0$. If $x_{t,i'}$ is increased, then $x_{t,i}$ is increased by the same amount, hence order constraints are verified. If $x_{t,i}$ is increased, it means that $val/flow > val'/flow'$. Hence, $val/flow > (val' - val)/(flow' - flow)$. Consequently, the algorithm will increase $x_{t,i'}$ only if $x_{t,i} = 1$, hence order constraints are verified. □

Theorem 1. *Algorithm 1 defines an optimistic heuristic.*

Proof. Let s be an integer solution for the 1-HUC, for which variables $x_{t,i}$ verify constraints (a) (b) (c) and (f). Let $\hat{s}$ be a fractional solution for the 1-HUC obtained with Algorithm 1, for which $\hat{x}_{t,i}$ verify all constraints (a), (b) (c), and constraint (f) only for time periods 1 to $t \leq T$. Consider $\hat{s}$ and s to be identical for time periods 1 to t.

Let $\mathcal{X}$ (resp. $\mathcal{Y}$) be the variables such that $x_{t',i} < \hat{x}_{t',i}$ $\forall x_{t',i} \in \mathcal{X}$ (resp. $x_{t',i} > \hat{x}_{t',i}$ $\forall x_{t',i} \in \mathcal{Y}$).

Clearly, for each variable $x_{t',i} \in \mathcal{X}$ of positive profitability, a fractional value for $x_{t',i}$ increases the value of the objective function compared to $x_{t',i} = 0$. Similarly, for each variable in $\mathcal{Y}$ of negative profitability, a fractional value for $x_{t',i}$ increases the value of the objective function compared to $x_{t',i} = 1$.

Let $\mathcal{X}^- \subseteq \mathcal{X}$ and $\mathcal{Y}^- \subseteq \mathcal{Y}$ be the variables with negative profitability. Suppose $|\mathcal{X}^-| > 0$. By construction of $\hat{s}$, the variables of $\mathcal{X}^-$ have value greater than 0 only in order to yield a feasible solution, with respect to a lower bound $\beta_{t'}$. In such a case the total flow of $\hat{s}$ from time period 1 to t' is exactly $\beta_{t'}$ by construction of $\hat{s}$. As solution s also verifies all lower bounds, we deduce $|\mathcal{Y}^-| > 0$. Otherwise there is either a contradiction with the construction of $\hat{s}$, or s does not verify all lower bounds. Hence, the total flow of s from time period 1 to t' is at least $\beta_{t'}$. By definition, s and $\hat{s}$ are identical from time period 1 to t, consequently the only difference is on time periods $t+1$ to t'. By construction of $\hat{s}$ the variables of $\mathcal{X}^-$ are the most profitable and have fractional value in $\hat{s}$. Consequently, their weighted value in the objective function must be higher than those of $\mathcal{Y}^-$ in the integer solution s.

A similar proof can be made for variables in $\mathcal{Y}^+ \subseteq \mathcal{Y}$ and $\mathcal{X}^+ \subseteq \mathcal{X}$ the variables with positive profitability.

The value of $\hat{s}$ is then greater or equal to the value of s. □

It is possible to tighten the fractional solution returned by Algorithm 1 while keeping it optimistic. Clearly, an integer solution is necessarily of a total flow which is a combination of the flows from the operating points. Therefore the flow of an integer solution is necessarily a multiple of the greatest common divisor (GCD) of the operating points' flows. When the heuristic increases the value of a variable, we can increase or reduce this value so that the total flow of the returned solution remains a multiple of the GCD relative to the operating points' flows. Note that since the flows are identical from one time period to another, we can quickly compute the GCD by considering only the flows of a single time period.

Algorithm 1 Algorithm OptimisticHeuristic

Require: A path p from time period 1 to t, a graph G^*_{KP}, GCD the GCD of the flow

Initialize a fractional solution $\hat{x}$ with all variables to 0
for $t' \in [1;T]$ and $i \in [1;M]$ **do**
  if $t' \leq t$ AND p requires operating point i at time period t' **then**
    $\hat{x}_{t',i} = 1$
  else
    $\hat{x}_{t',i} = 0$
  end if
end for
Initialize a list $L = []$
for $t' \in [t+1;T]$ and $i \in [1;M]$ **do**
  $flow \leftarrow \sum_{i'=1}^{i} D_{i'}$
  val the value of an arc towards (t', i) in G^*_{KP}
  add $(t', i, val, flow)$ in L, sorted by decreasing $val/flow$
end for
while $\exists t' \in [t+1, T]$ such that $\hat{x}$ does not verify $\beta_{t'}$ OR exists quadruplet in L with $val > 0$ **do**
  $(t'', i, val, flow) \leftarrow$ first in L with $t'' \leq t'$
  if $val \leq 0$ **then**
    set $\hat{x}_{t'',i}$ to minimum such that $\beta_{t'}$ verified and $x_{t'',i} \cdot flow \mod GCD = 0$; if $\beta_{t'}$ cannot be verified $\hat{x}_{t'',i} \leftarrow 1$
  else
    set $\hat{x}_{t'',i}$ to maximum such that all upper bounds are verified and $x_{t'',i} \cdot flow \% GCD = 0$
  end if
  for $i' \in [1;i]$ **do**
    $\hat{x}_{t'',i'} = \max(\hat{x}_{t'',i'}, \hat{x}_{t'',i})$
  end for
  for $(t'', i', val', flow') \in L$ with $i' \in]i;M]$ **do**
    $val' \leftarrow val' - val$
    $flow' \leftarrow flow' - flow$
  end for
  for $(t'', i, val, flow) \in L$ such that $\hat{x}_{t'',i}$ cannot be increased **do**
    remove $(t'', i, val, flow)$ from L
  end for
end while
return the value of $\hat{x}$

5.3 HA* algorithm

For a 1-HUC with an objective function to maximize, the principle of HA* is the following. Consider a pool of partial solutions evaluated with the heuristic. At each iteration, the partial solution with the highest heuristic value is considered and removed from the pool. From the partial solution considered, we complement it by adding neighbors relative to its last vertex. Once a solution is found, its value is used as a bound to remove some more partial solutions from the pool. Indeed, if the solution's value is higher than a partial solution's heuristic value, then the partial solution can be removed from the pool. Once the pool of partial solutions is empty, the algorithm stops and the best solution found is the optimal solution.

We underline the need of a tight optimistic heuristic. If the heuristic is not optimistic, or optimistic but too loose, there are fewer cases where one can prune partial solutions while guaranteeing optimality. Hence, more vertices are developed which can exponentially increases the computational time.

For readability purposes, we introduce three structures:

Definition 5 (Path structure)**.** The path structure has three attributes: *vertices* the list of vertices of the path; *val* the value of the path with respect to the objective function; *heur* the optimistic heuristic value.

Definition 6 (Vertex structure)**.** A vertex structure has two attributes: t the time period and d the cumulated flow $\mathcal{D}_{1,t}$, as defined for the vertices of G^*_{KP} in Section 4.

Definition 7 (Arc structure)**.** The arc structure has one attribute: *val* the value as defined for the arcs of G^*_{KP} in Section 4.

Algorithm 2 presents the pseudo-code of HA*, using the three previously described structures as well as $OptimisticHeuristic$. The considered graph is G^*_{KP}, which is G_{KP} as illustrated in Fig. 2a with bounds β^*_t and α^*_t. The dominance rules used in Algorithm 2 are the dominance rules 1 and 2.

6 Experimental results

Results are computed on a single thread of an Intel(R) Core(TM) i7-9850H CPU @ 2.60GHz processor, with 2 CPUs of 8 cores, with Linux as operating system. All algorithms are developed with C++. Version 12.8 of CPLEX with default setting is used to solve the MILP formulation F_{1HUC}.

6.1 Instances

From a large set of realistic instances derived from a real EDF plant, a first set A of 13 instances is obtained. These 13 instances are retained as preliminary results have shown that formulation F_{1HUC} is not trivially solved. This emphasizes the need of an efficient alternative in these cases. Table 2 depicts for

Algorithm 2 Algorithm HA*

Require: A graph G^*_{KP}
  Initialize a path p as follows: $p.vertices = \{(0,0)\}$, $p.val = 0.0$, $p.heur = OptimisticHeuristic(p)$
  Initialize a list of paths $L_p = [p]$
  Initialize the value of the best solution $bestVal = -\infty$
  while L_p not empty **do**
    $p \leftarrow$ first path in L_p
    remove p from L_p
    $v \leftarrow$ last vertex of $p.vertices$
    for arc a from v to u **do**
      $q.vertices = p.vertices \cup u$, $q.val = p.val + a.val$, $q.heur = OptimisticHeuristic(q)$
      if $|q.vertices| = T + 1$ **then**
        $bestVal \leftarrow \max(bestVal, q.val)$
        remove $q' \in L_p$ with $q'.val + q'.heur \leq bestVal$
      else
        $dom \leftarrow FALSE$
        for $q' \in L_p$ **do**
          if q' dominates q **then**
            $dom \leftarrow TRUE$
          end if
          if q dominates q' **then**
            remove q' from L_p
          end if
        end for
        if $dom = FALSE$ and $q.val + q.heur > bestVal$ **then**
          add q in L_p by keeping L_p sorted by decreasing $q.val + q.heur$
        end if
      end if
    end for
  end while
  return the solution of value $bestVal$

each instance the main characteristics, namely the number of time periods, the number of operating points and the presence of a constraining minimum (resp. maximum) bound on the volume at the last time period. The instances cover three cases, namely when there is only an upper bound, only a lower bound, or a target volume with a constraining upper and lower bound. In the case of an equality constraint, the instance becomes infeasible, which is out of the scope of the instances considered in this paper. Hence, the target volumes are not equality constraints, but rather tight window constraint. For instances with a target volume, more resource window constraints are obtained by propagating the bounds β^* and α^* from the bounds at the last time period to the previous ones.

The water flows D and powers P are in the order of 10^3 to 10^4, with volumes in the order of 10^7. For target volumes, the difference between the upper and lower bound is in the order of 10^3, which is small enough with respect to the flows to yield very few vertices at time period T in G^*_{KP}.

Table 2: Main characteristics of instances set A

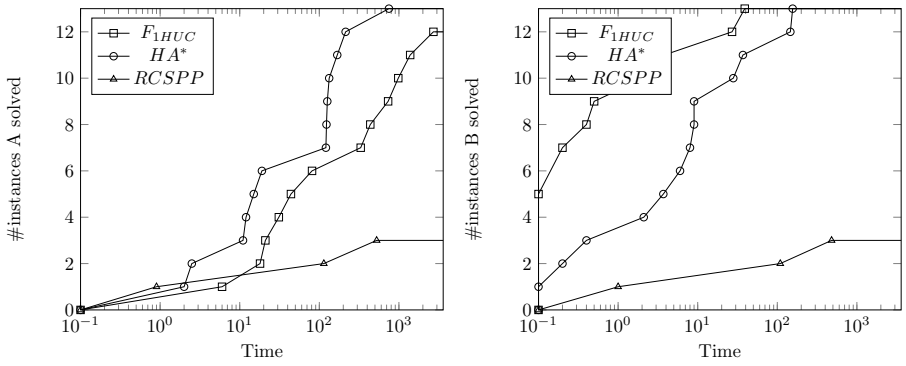
instance	T	M	minimum volume	maximum volume
1	96	4	✗	✓
2	96	4	✓	✗
3	96	4	✓	✓
4	96	7	✗	✓
5	96	7	✓	✗
6	96	7	✓	✓
7	96	8	✗	✓
8	96	8	✓	✓
9	96	15	✗	✓
10	96	17	✗	✓
11	96	18	✓	✗
12	96	21	✗	✓
13	96	18	✗	✓

A second set B of 13 instances, similar to the first set, is also constructed. The only differences are bounds β_T and α_T that are shifted as follows. Let an instance with bounds β^A_T and α^A_T be in set A. A random value $k \in [-9999; -1000] \cup [1000; 9999]$ is chosen. Bounds of an instance for set B are $\beta^B_T = \beta^A_T + k$ and $\alpha^B_T = \alpha^A_T + k$. Note that this shift is very small for these instances. Indeed, the water flows can be in the order of 10^4, there are nearly 100 time periods and the cumulated flow $\mathcal{D}_{1,T}$ is in the order of 10^6. The shift k is at most 1% of $\overline{\mathcal{D}}_{1,T}$. As shown in the following results, slightly modifying an instance can drastically impact the computational time.

6.2 Experimental results

In order to benchmark HA*, all instances are solved with HA* as well as with two alternative methods. The first alternative is a classical RCSPP algorithm [12] adapted to the 1-HUC [11]. The second alternative is to use CPLEX to solve

F_{1HUC} described in Section 3. A third alternative is solving F^{+}_{1HUC}, which is formulation F_{1HUC} described in Section 3 with tighter bounds described in Section 5.1. However, solving F^{+}_{1HUC} leads to very similar results to F_{1HUC}, hence results for F^{+}_{1HUC} are not included in the following. All algorithms use a single thread, with a time limit of one hour.



(a) Total number of instances A solved per computational time

(b) Total number of instances B solved per computational time

Figure 3: Total number of instances A and B solved per computational time

Figures 3a and 3b represent the number of instances solved by each algorithm with respect to the computational time. Clearly, for instance set A, HA* is the most efficient alternative. Indeed, it solves every instance and requires less time than the other alternatives. For instance set B, solving F_{1HUC} is the most efficient alternative. Note that the difference between the solving times of F_{1HUC} and HA* is only of a few seconds for instance set B, as most instances are solved in less than 10 seconds with HA*. Solving F_{1HUC} is the least robust alternative when it comes to computational times. Indeed, when comparing the results between instance sets A and B, the computational times are drastically different with F_{1HUC}, whereas for HA* there is a smaller difference, and for the RCSPP algorithm the results are the same. The RCSPP algorithm fails to solve 10 out of 13 instances, for both instance sets A and B. This shows that the RCSPP algorithm is inefficient at solving the 1-HUC. We further explain the results in the following, by introducing Tables 3 and 4 with detailed results.

Tables 3 and 4 give, for each instance, the value of the objective function and the computational times obtained for all algorithms, as well as the optimality gap and the number of Branch & Bound nodes returned by the MILP solver. If the MILP solver proves optimality, the gap is noted "opt". When the time limit is reached, the time is noted "-". Results are presented for each instance individually, as well as the average (avg) and the standard deviation (sd) for the solved instances. When the time limit is reached, a computational time of 3600 seconds is accounted for in the average and the standard deviation. The computational time of the most efficient algorithm is emphasized in bold for

each instance, as well as the average and the standard deviation for each set of instances.

Table 3: Performance of solving F_{1HUC}, the RCSPP algorithm and HA* on instance set A

	F_{1HUC}				RCSPP		HA*	
instance	value	#nodes	gap	time	value	time	value	time
1	−25 428.80	10 232 857	opt	2713.4	-	-	−25 428.80	**2.5**
2	43 010.90	2 591 995	opt	437.0	43 010.9	**0.9**	43 010.90	2.3
3	3556.54	5 034 351	opt	1384.8	-	-	3556.54	**12.0**
4	2462.90	167 490	opt	44.1	-	-	2462.90	**11.1**
5	111 115.00	101 570	opt	**17.8**	111 115.0	525.2	111 115.00	18.8
6	−1706.62	888 735	opt	331.4	-	-	−1706.62	**14.9**
7	5692.65	115 626	opt	**6.2**	-	-	5692.65	120.6
8	16 581.10	487 218	opt	**30.9**	-	-	16 581.10	126.0
9	−71 645.90	176 123	opt	**21.4**	-	-	−71 645.90	133.0
10	−525 446.00	901 533	opt	732.1	-	-	−525 446.00	**168.4**
11	44 421.20	1 535 139	opt	982.0	44 421.2	**114.1**	44 421.20	213.9
12	−20 329.90	1 624 614	0.58	-	-	-	−20 324.00	**748.8**
13	−103 435.00	365 253	opt	**80.7**	-	-	−103 435.00	122.8
avg	-	1 865 577	0.58	798.6	-	1431.0	-	**130.4**
sd	-	2 755 062	0.0	1101.1	-	2818.4	-	**191.7**

Table 4: Performance of solving F_{1HUC}, the RCSPP algorithm and HA* on instance set B

	F_{1HUC}				RCSPP		HA*	
instance	value	#nodes	gap	time	value	time	value	time
1	−25 430.30	873	opt	**0.2**	-	-	−25 430.30	0.4
2	42 993.00	0	opt	**0.0**	42 993.00	1.0	42 993.00	**0.0**
3	3700.23	11	opt	**0.0**	-	-	3700.23	0.2
4	2462.90	16 330	opt	**2.1**	-	-	2462.90	3.7
5	111 143.00	34 982	opt	**3.8**	111 143.00	481.5	111 143.00	8.4
6	−1460.33	2519	opt	**0.4**	-	-	−1460.33	2.1
7	5936.31	522 284	opt	**39.2**	-	-	5936.31	155.7
8	16 730.40	6403	opt	**0.5**	-	-	16 730.40	8.6
9	−132 290.00	0	opt	**0.0**	-	-	−132 290.00	9.2
10	−525 246.00	46 425	opt	**27.1**	-	-	−525 246.00	37.2
11	44 646.80	0	opt	**0.0**	44 646.80	109.1	44 646.80	6.2
12	−20 153.10	1586	opt	**0.1**	-	-	−20 153.10	146.3
13	−103 339.00	2314	opt	**0.2**	-	-	−103 339.00	27.7
avg	-	48 748	-	**5.7**	-	1437.1	-	31.2
sd	-	137 446	-	**12.0**	-	2814.7	-	52.2

There is a clear difference in computational time between set A and B. Indeed, F_{1HUC} is solved for 12 out of the 13 instances of set A, and needs between 6 and 2713 seconds, while it is solved for all instances of set B, most of them instantaneously. Similarly, HA* solves all instances of set A and needs between 2 and 748 seconds, while it solves all instances of set B in less than 156 seconds. Note however that there is no noticeable computational time difference between solving instance set A and B with the RCSPP algorithm.

The RCSPP algorithm is only able to solve instances 2, 5 and 11, for both sets. This is because for any other instance, the maximum volume of the up-

stream reservoir is constrained at the last time period. In this case, the value β_T becomes positive, meaning that there is a minimum bound on the resource. As mentioned in [3], when there is a minimum bound on the resource, the dominance properties of the RCSPP algorithm cannot always be applied, thus leading to large computational times. Clearly, HA* outperforms the RCSPP algorithm. Even when there is no minimum bound on the resource, the RCSPP algorithm yields larger computational times for all instances of set B and instance 5 of set A.

When comparing HA* algorithm to solving F_{1HUC} on the most difficult instances, the former outperforms the latter in terms of computational time and number of instances solved. On the one hand, HA* is more stable with respect to the computational times. Indeed, HA* only requires more than 10 minutes once (instance 12 of set A), while solving F_{1HUC} requires more than 10 minutes for 5 of the 26 instances (instances 1, 3, 10, 11 and 12 of set A). Moreover, F_{1HUC} is not solved to optimality for instance 12 of set A, and the best value found is not the optimal value obtained with HA*. On the other hand, solving F_{1HUC} appears to be more efficient on easier instances than HA*. In this case, there are numerous instances where the difference between the two approaches is within a few seconds (instances 1, 2, 3, 4 and 6 of set B).

The stability with respect to the computational time is noticeable on the average and standard deviation. Indeed, the average time difference between set A and B is much smaller for HA* than for the MILP solver. The standard deviation for HA* is much smaller on set A, and slightly higher for set B when compared to solving F_{1HUC}. Besides, one can compute the total average and total deviation of the resolution time for all 26 instances. The total average time and standard deviation are 402.1 seconds and 873.7 seconds for the MILP solver, and 80.3 seconds and 149.0 seconds for HA*.

7 Conclusion

In this paper, the HA* algorithm is proposed as an exact variant of the A* algorithm dedicated to the 1-HUC. It has been adapted through a dedicated optimistic heuristic and a bound tightening pre-processing, propagating lower and upper bounds from any time period to another. On a set of realistic instances, algorithm HA* is shown to be both more stable and more efficient on average in terms of computational times compared to solving F_{1HUC}. Also, HA* outperforms the standard labeling algorithm for the RCSPP.

A natural direction for a future work would be to extend the proposed algorithm in order to take into account additional constraints of the 1-HUC. A promising perspective is to include the HA* algorithm in a decomposition of a hydraulic valley with multiple plants. Beyond the HUC, another interesting perspective could also be to try and generalize such an algorithm for other problems with window resource constraints.

References

[1] A. Renaud, "Daily generation management at Electricité de France: from planning towards real time," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1080–1093, 1993.

[2] G. Hechme-Doukopoulos, S. Brignol-Charousset, J. Malick, and C. Lemaréchal, "The short-term electricity production management problem at EDF," *Optima Newsletter*, vol. 84, pp. 2–6, 2010.

[3] W. van Ackooij, C. d'Ambrosio, D. Thomopulos, and R. S. Trindade, "Decomposition and shortest path problem formulation for solving the hydro unit commitment and scheduling in a hydro valley," *European Journal of Operational Research*, vol. 291, no. 3, pp. 935–943, 2021.

[4] G. Ardizzon, G. Cavazzini, and G. Pavesi, "A new generation of small hydro and pumped-hydro power plants: Advances and future challenges," *Renewable and Sustainable Energy Reviews*, vol. 31, pp. 746–761, 2014.

[5] Y. Sahraoui, P. Bendotti, and C. d'Ambrosio, "Real-world hydro-power unit-commitment: Dealing with numerical errors and feasibility issues," *Energy*, vol. 184, pp. 91–104, 2019.

[6] J. I. Pérez-Díaz, J. R. Wilhelmi, and L. A. Arévalo, "Optimal short-term operation schedule of a hydropower plant in a competitive electricity market," *Energy Conversion and Management*, vol. 51, no. 12, pp. 2955–2966, 2010.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[8] W. Fan, X. Guan, and Q. Zhai, "A new method for unit commitment with ramping constraints," *Electric Power Systems Research*, vol. 62, no. 3, pp. 215–224, 2002.

[9] R. Taktak and C. D'Ambrosio, "An overview on mathematical programming approaches for the deterministic unit commitment problem in hydro valleys," *Energy Systems*, vol. 8, no. 1, pp. 57–79, 2017.

[10] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[11] W. van Ackooij, C. d'Ambrosio, L. Liberti, R. Taktak, D. Thomopulos, and S. Toubaline, "Shortest path problem variants for the hydro unit commitment problem," *Electronic Notes in Discrete Mathematics*, vol. 69, pp. 309–316, 2018.

[12] C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. Marathe, and D. Wagner, "Engineering label-constrained shortest-path algorithms," in *Algorithmic Aspects in Information and Management: 4th International Conference, AAIM 2008, Shanghai, China, June 23-25, 2008. Proceedings 4*. Springer, 2008, pp. 27–37.

[13] A. Arce, T. Ohishi, and S. Soares, "Optimal dispatch of generating units of the Itaipú hydroelectric plant," *IEEE Transactions on power systems*, vol. 17, no. 1, pp. 154–158, 2002.

[14] C.-T. Cheng, S.-L. Liao, Z.-T. Tang, and M.-Y. Zhao, "Comparison of particle swarm optimization and dynamic programming for large scale hydro unit load dispatch," *Energy Conversion and Management*, vol. 50, no. 12, pp. 3007–3014, 2009.

[15] M. Kruber, A. Parmentier, and P. Benchimol, "Resource constrained shortest path algorithm for EDF short-term thermal production planning problem," 2018. [Online]. Available: https://arxiv.org/abs/1809.00548

[16] L. Turner, "Variants of the shortest path problem," *Algorithmic Operations Research*, vol. 6, no. 2, pp. 91–104, 2011.

[17] C. C. Ribeiro and M. Minoux, "A heuristic approach to hard constrained shortest path problems," *Discrete Applied Mathematics*, vol. 10, no. 2, pp. 125–137, 1985.

[18] J. E. Beasley and N. Christofides, "An algorithm for the resource constrained shortest path problem," *Networks*, vol. 19, no. 4, pp. 379–394, 1989.

[19] X. Zhu and W. E. Wilhelm, "Three-stage approaches for optimizing some variations of the resource constrained shortest-path sub-problem in a column generation context," *European journal of operational research*, vol. 183, no. 2, pp. 564–577, 2007.

[20] ——, "A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation," *Computers & Operations Research*, vol. 39, no. 2, pp. 164–178, 2012.