Hierarchical Constraint-Based Scheduling for Multi-Robot Observation Missions

Paper 5731

Abstract

In this paper, we consider scheduling problems involving tasks (*area observations*) that can be realized by several resources (*robots*), together with setup operations between tasks (*robot moves between observations*) that are decomposed into low level operations (*moves on edges of a waypoint graph through multiple possible paths*). To deal with such problems, we introduce a framework for representing hierarchical scheduling problems, and for which constraint programming models can be automatically generated. We also define an iterative two-layer constraint-based decision process that alternates between the fast synthesis of high-level schedules based on a coarse-grain model of setup operations, and the production of a detailed schedule based on a fine-grain model of these operations. Experiments realized on representative benchmarks of a multi-robot application show the efficiency of the approach.

1 Introduction

In this paper, we deal with so-called *hierarchical scheduling problems*, that have a hierarchical structure in multiple ways. To present our approach, we focus on one of the real case-studies that motivates this work, namely a multi-robot deployment mission in which a fleet of robots must perform a set of observations on specific areas of a field, for situation awareness issues. Our problem is to allocate each candidate observation to a robot, schedule the sequence of observation tasks realized by each robot, and plan navigation tasks between observation locations. The objective is to perform all observations as quickly as possible. It is assumed that robots cannot perform more than one observation at a time. They must also transfer observation data in real time to the mission center, and for this purpose each robot uses a specific emission frequency. To avoid interferences, two robots that use the same frequency cannot transfer observation data simultaneously. Redundancy is also useful in this kind of application, therefore each observation target must be observed by several different robots. Moreover, some precedence constraints can be imposed over observations. Finally, a graph of waypoints is used to represent the structure of the field, and the movements of robots between observation locations can be broken down into successive movements between pairs of adjacent waypoints. To avoid collisions, each link between two waypoints cannot be occupied by more than one robot at a time.

These specifications lead to a task scheduling problem that includes (1) *tasks which can be realized by several candidate resources*, (2) *setup operations* between the tasks realized by a resource (corresponding to the moves of robots between observation locations), (3) *several alternatives for realizing each setup operation* (several candidate navigation paths between two distant observations), and (4) *for each setup alternative, a decomposition into low level tasks* (in our case, each path traversal is decomposed into a set of moves on edges of the waypoint graph).

The first contribution of this paper is that to deal with such applications in a generic way, we introduce a framework for representing Hierarchical Scheduling Problems (HSPs). This framework is inspired by *Hierarchical Task Networks* (HTNs (Erol, Hendler, and Nau 1994)) used in the planning community. The main difference is that HSPs define scheduling problems based on tasks and resources, while HTNs are basically suitable for planning problems based on actions and states. We also introduce techniques to automatically generate *flat* constraint programming encodings from HSPs. For non-expert in Constraint Programming (CP), this allows to benefit both from a hierarchical view of the domain in the form of HSPs and from the strength of CP techniques.

The second contribution is that to deal with HSPs having a large number of candidate task decompositions, we define a hierarchical decision strategy that splits the decision process into several scheduling layers. Basically, we divide the process into (1) layer L1 that produces high-level decisions based on a coarse-grain model of all setup operations, and (2) layer L2 that produces detailed schedules based on a fine-grain model of all low-level tasks realized during setups. We then iteratively solve the scheduling problems of layers L1 and L2, given that at each step the planning data used by L1 (such as the approximate durations of setups) is updated according to the detailed schedules returned by L2.

The paper is organized as follows. Section 2 describes the HSP framework and the proposed CP encoding. Section 3 defines the iterative two-layer decision process. Section 4 provides experimental results obtained on representative benchmarks of the robotic application. Section 5 discusses related works, and Section 6 gives some perspectives.

2 Hierarchical Scheduling Problems (HSPs)

2.1 General Scheduling Model

We consider a set of disjunctive resources $\mathcal{R}$, which cannot be used by several tasks simultaneously. This set is partitioned between the set $\mathcal{R}^d$ of simple disjunctive resources and the set $\mathcal{R}^s$ of disjunctive resources with setup times, for which a transition duration is required between successive tasks realized by the resource. Each resource $r \in \mathcal{R}^s$ has S possible running states and each task consuming r requires a particular resource state $s \in [1..S]$. For every pair of resource states $(s, s') \in [1..S]^2$, a minimum setup duration function setup_r gives the duration $\mathit{setup}_r(s, s') \in \mathbb{N}$ required between the end of a task using r in state s and the start of a task using r in state s'.

Resources in $\mathcal{R}$ are used to execute *tasks*. A task t is defined by:

- a release date rd_t after which it can start and a due date dd_t before which it must end;
- the set of resources $R_t \subseteq \mathcal{R}$ it consumes all along its execution (from the start of the task to its end), with for every resource with setup $r \in R_t \cap \mathcal{R}^s$ the state $st_{t,r}$ required for r during the execution of t.

A task is either *primitive* or *compound*:

- a *primitive task* o, also called an *operation*, has a fixed duration du_o;
- a *compound task* c has a list of possible *decomposition methods* $M_c = [m_{c,1}, \dots, m_{c,k}]$ usable for realizing c; each decomposition method $m \in M_c$ corresponds to a so-called *task network* (T_m, Ψ_m).

Formally, a task network (T, Ψ) is composed of a set of tasks T and a set of constraints Ψ over these tasks. We consider two types of constraints in Ψ:

- *temporal constraints* over tasks in T, denoted Ψ_{temp}; in this paper, we only consider a set of acyclic precedence constraints between pairs of tasks, but the approach can deal with any minimum and maximum distance constraint between the start and end time-points of tasks in T;
- *decomposition constraints*, denoted Ψ_{decomp}, that restrict the possible combinations of decompositions of compound tasks in T; they are expressed over $\{\mathbf{mth}(c) \mid c \in T \text{ s.t. } c \text{ is compound}\}$, where $\mathbf{mth}(c)$ denotes the index i of the method $m_{c,i}$ chosen for realizing task c.

A *Hierarchical Scheduling Problem* (HSP) is then defined by a *task network* (T_0, Ψ_0), called the *root task network*, which represents the set of high-level tasks to be realized. In the following, we denote by $\mathcal{O}, \mathcal{C}$, and $\mathcal{M}$ respectively the set of all operations, compound tasks, and methods involved in the full hierarchical decomposition of the root task network. Moreover, for any resource $r \in \mathcal{R}$, $\mathcal{T}_r$ denotes the set of tasks t (primitive or compound) that consume resource r, *i.e.* such as $r \in R_t$.

To illustrate the previous definitions, we consider an HSP associated with the placement of embedded functions over a multi-core platform. Consider two functions F_A and F_B that can be allocated to four distinct cores $c_1 \dots c_4$, and such that F_A must transfer a data D_{AB} to F_B. If F_A and F_B are placed over the same core, this transfer takes 1 time unit, otherwise it uses the embedded network and takes 10 time units. Each core is viewed as a disjunctive resource, as well as the network between cores. Fig. 1 provides a graphical description of the hierarchical problem that can be built for this simple problem (real instances involve thousands of functions and data transfers). In this problem, the root task network (T_0, Ψ_0) contains three compound tasks $(T_0 = \{F_A, D_{AB}, F_B\})$ and two precedence constraints $F_A \rightarrow D_{AB}$ and $D_{AB} \rightarrow F_B$. Task F_A has four decomposition methods $M_{F_A} = [m_{F_A,1}, \dots, m_{F_A,4}]$, where the ith method corresponds to a computation of the function on the ith core. Method $m_{F_A,i}$ points to a task network $(T, \Psi) = (\{F_A^i\}, \emptyset)$ which contains a unique primitive task F_A^i of fixed duration that consumes core c_i. The modeling is similar for F_B. Data transfer task D_{AB} has two possible decomposition methods. The first one points to a task network reduced to a single primitive task D_{AB}^{core} of duration 1 and corresponds to a data transfer over the same core. The second one also points to a task network reduced to a single primitive task D_{AB}^{net} of duration 10 which is used for a data transfer over the network, and which consumes the network resource. At the level of the root task network (T_0, Ψ_0), one decomposition constraint is added to Ψ_0 to express that a data transfer over the network is used when the two functions are allocated to distinct cores $((\mathbf{mth}(D_{AB}) = 2) \leftrightarrow (\mathbf{mth}(F_A) \neq \mathbf{mth}(F_B)))$.

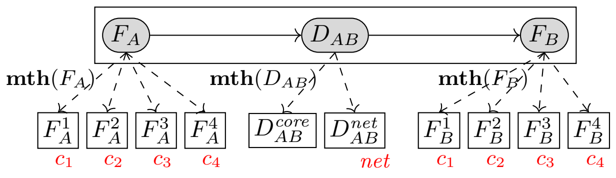

Figure 1: HSP for the deployment of embedded functions over a multi-core platform

Additional assumptions: we assume that the HSPs addressed are *well formed*. More precisely, it is assumed that:

- each task belongs to a unique task network and the total number of tasks in an HSP is finite; this means that the set of candidate decompositions of the root tasks is finite;
- given a task t consuming a resource with setup times r, the release date rd_t of t is assumed to be greater than or equal to the setup duration required to put resource r in state $st_{t,r}$ from the beginning of the schedule.

2.2 Encoding in Constraint Programming

Given an HSP, it is possible to generate a *flat* CP encoding for minimizing the makespan. In this encoding, each task is represented by an *interval variable*. For each interval variable itv, $\mathbf{start}(itv)$, $\mathbf{end}(itv)$ and $\mathbf{pres}(itv)$ respectively denote the start date, the end date and the presence of the interval in the solution. Then, we introduce the following variables:

- for each task $t \in \mathcal{O} \cup \mathcal{C}$, an interval variable $itv_t \in [rd_t..dd_t]$;
- for each compound task $c \in \mathcal{C}$, an integer variable $mth_c \in [0..|M_c|]$ that represents the index of the decomposition method chosen for realizing c (value 0 used when the task is not present);
- for each compound task $c \in \mathcal{C}$ and each decomposition method $m \in M_c$, an interval variable $itv_m \in [rd_c..dd_c]$.

In the CP encoding, we define Constraints (1) to (8) given below, which use several constraints available in the CpOptimizer tool.[1] They express that all root tasks must be executed (1) and that every compound task is realized using exactly one of its decomposition methods (2). The latter uses the *alternative*(itv, I) constraint available in CpOptimizer, which specifies that an interval itv is equal to exactly one of the intervals in I. Also, each decomposition interval must cover all its subtasks (3). The latter uses the *span* constraint specifying that an interval must cover all present intervals belonging to a given set. The presence of subtasks of a method must also coincide with the presence of the method (4). Constraints (1)-(4) actually correspond to already existing CP encodings of work breakdown structures.[2]

The other constraints introduced are used to represent the scheduling and decomposition constraints of the problem. First, every operation has a fixed duration (5). Also, if a compound task is not present in a solution, then the method choice index for this task is set to its default value (6), otherwise this index is consistent with the presence of the possible decomposition intervals (7). Constraints (8)-(9) enforce that tasks realized by disjunctive resources must not overlap, taking into account setup durations for resources with setup operations. For every task network (T, Ψ) used in the model, all precedence constraints in Ψ_{temp} must be satisfied (11). The latter uses the *endBeforeStart* constraint of CpOptimizer, which imposes the end of a first interval to precede the start of a second. Last, other decomposition constraints $\psi \in \Psi_{decomp}$ must be satisfied (11). For this, we directly write constraint ψ conditioned by the presence of the decomposition method from which the task network comes (or unconditioned for the root task network). For the experiments, decomposition constraints are manually added to the CP model generated, but there is no technical difficulty to integrate them in the generation procedure.

$$\forall t \in T_0,\ \mathbf{pres}(itv_t) = 1 \quad (1)$$
$$\forall c \in \mathcal{C},\ alternative(itv_c, \{itv_m \mid m \in M_c\}) \quad (2)$$
$$\forall m \in \mathcal{M},\ span(itv_m, \{itv_t \mid t \in T_m\}) \quad (3)$$
$$\forall m \in \mathcal{M}, \forall t \in T_m,\ \mathbf{pres}(itv_m) = \mathbf{pres}(itv_t) \quad (4)$$
$$\forall o \in \mathcal{O},\ duration(itv_o, du_o) \quad (5)$$
$$\forall c \in \mathcal{C},\ (\mathbf{pres}(itv_c) = 0) \leftrightarrow (mth_c = 0) \quad (6)$$
$$\forall c \in \mathcal{C}, \forall k \in [1..|M_c|],\ \mathbf{pres}(itv_{m_{c,k}}) = (mth_c = k) \quad (7)$$
$$\forall r \in \mathcal{R}^d,\ noOverlap(\{itv_t \mid t \in \mathcal{T}_r\}) \quad (8)$$
$$\forall r \in \mathcal{R}^s,\ noOverlap(\{(itv_\tau, st_{t,r}) \mid t \in \mathcal{T}_r\}, setup_r) \quad (9)$$
$$\text{for every task network } (T, \Psi), \forall \psi = (t, t') \in \Psi_{temp},\ endBeforeStart(itv_t, itv_{t'}) \quad (10)$$
$$\text{for every task network } (T, \Psi), \text{ constraints in } \Psi_{decomp} \quad (11)$$

If there is a unique decomposition method m for a compound task c, intervals itv_m and itv_c are equal. In this case, only one interval instance is defined and Constraint (2) is not generated. Similarly, if a decomposition method m has a unique subtask t (*i.e.* $T_m = \{t\}$), intervals itv_m and itv_t are equal. In this case, only one interval instance is defined and Constraints (3)-(4) are not generated.

When the goal is to minimize the makespan, the objective function generated consists in minimizing the maximum end time of a root task, that is to minimize $\max_{t \in T_0} \mathbf{end}(itv_t)$.

Note that the model and the encoding can easily be generalized to cumulative resources having a finite capacity or to non renewable resources. This allows to deal with *Resource Constrained Project Scheduling Problems (RCPSPs (Brucker et al. 1999)) with modes*, and more generally with kinds of hierarchical RCPSPs. In the end, the representation framework obtained is both simple and easy to use by non-CP experts, and quite expressive.

3 Iterative Hierarchical Decision Strategy

We now come back to the robotic application presented in the introduction. To tackle this problem, a first option is to define a unique HSP containing, for each robot, all potential setup operations that might be used between any two observation tasks, all possible decompositions of these potential setup operations, and all primitive tasks involved in these decompositions. For our application, this leads to a number of tasks in $\Theta(RO^2PL)$ with R the number of robots, O the number of observations, P the maximum number of candidate paths between two observations, and L the maximum number of network links on a single path.

To avoid handling such a huge number of tasks, another approach is to explicitly break down the problem to be solved into several sub-problems and to define a dedicated solution technique for each of these sub-problems. In this direction, we can use for our applications a two-layer decision strategy that first synthesizes a high-level schedule based on a coarse-grain model of setup operations (decision layer L1, which approximates these operations by simple setup durations), and then details this schedule based on a fine-grain model (decision layer L2, which only considers setup operations that are actually used in the coarse-grain solution). Such a top-down approach is quite commonly used in practice for hierarchical decision making. But as high-level decisions are computed from a coarse-grain model, it can fail to reach the highest quality solutions.

This is why we propose to iteratively use the two scheduling layers. More precisely, each time a new detailed schedule is produced by layer L2, input data of the imperfect coarse-grain model of layer L1 are updated, and a new high-level solution is looked for. Doing so, layer L1 iteratively learns a better approximation of the content of layer L2, the goal being to converge towards better full solutions. Before detailing these mechanisms, we first use the hierarchical

[1] https://www.ibm.com/analytics/cplex-cp-optimizer

[2] http://gdrro.lip6.fr/sites/default/files/JourneeROIA-IBM.pdf

scheduling framework introduced in the previous section for defining the models of decision layers L1 and L2.

3.1 High-level Scheduling Model: Layer L1

Inputs The inputs of the first layer of the multi-robot observation mission are:

- a set of *frequencies* $\mathcal{F} = [1..|\mathcal{F}|]$ for transferring observation data;
- a set of *robots* $\mathcal{R}ob = [1..|\mathcal{R}ob|]$; $\forall r \in \mathcal{R}ob$, $freq_r \in \mathcal{F}$ is the frequency used by r to emit data during observations;
- a set of *observation requests* $\mathcal{R}eq = [1..|\mathcal{R}eq|]$, corresponding to areas of the field that must be observed. For each request $j \in \mathcal{R}eq$, $duReq_j \in \mathbb{N}$ is the duration required to observe the area;
- a set of *(atomic) observations* $\mathcal{O}bs = [1..|\mathcal{O}bs|]$ to be performed. For each request $j \in \mathcal{R}eq$, it is required that different robots observe the corresponding area (redundancy). For each observation $i \in \mathcal{O}bs$, $req_i \in \mathcal{R}eq$ denotes the request associated with i;
- a set of acyclic precedence constraints $\mathcal{P} \subseteq \mathcal{R}eq^2$ between requests of $\mathcal{R}eq$;
- for each robot $r \in \mathcal{R}ob$, $duM_r : \mathcal{O}bs \times \mathcal{O}bs \rightarrow \mathbb{N}$ denotes the function such that for each robot r, $duM_r(i, i')$ gives the shortest duration required to move from the location of observation i to the location of observation i' over all possible paths of the waypoint network; we extend duM_r so that $duM_r(0, i')$ gives the shortest duration required to move from the initial location of r to observation i';
- a temporal horizon $H \in \mathbb{N}$ for the whole mission.

Hierarchical scheduling problem The HSP built for layer L1 is illustrated in Fig. 2. It contains two kinds of resources: the frequency resources (disjunctive), used to emit observation data, and the robot position resources (disjunctive with setup times) for which the states are the set of observations $[1..|\mathcal{O}bs|]$ and for each robot r, the setup duration function is duM_r. The root task network (T_0, Ψ_0) contains one compound task RQ_j per request $j \in \mathcal{R}eq$, and the set of precedence constraints $\mathcal{P}$ that hold over the requests. Each compound task RQ_j has a unique decomposition method. The latter defines a task network which contains one compound task O_i per observation $i \in \mathcal{O}bs$ such that $req_i = j$. Each compound task O_i has as many decomposition methods as the number of robots, and each of these decomposition methods defines a task network which contains a unique operation $O_{i,r}$ representing the realization of observation i by robot r. Task $O_{i,r}$ consumes one frequency resource (the frequency used by r), and it uses the position resource of robot r in state i. Its duration is given by the duration of request j. In this particular case, resources are consumed only by operations, and not by compound tasks. Release dates are determined based on the duration required by each robot to go from its initial location to every observation area. The due date of all tasks is set up to temporal horizon H. Last, at the level of the task network associated with the decomposition of a request j, a decomposition constraint is added to ensure that observations associated with j are realized by distinct robots (e.g., $\mathbf{mth}(O_1) \neq \mathbf{mth}(O_2)$ for the example given in Fig. 2, and more generally an *alldiff* constraint between the indices of the chosen decomposition methods). As explained in the previous section, a CP model can be directly generated from this hierarchical description.

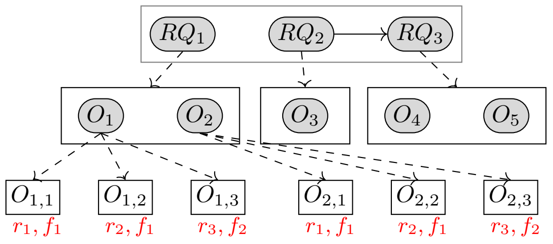

Figure 2: HSP for layer L1

3.2 Low-level Scheduling Model: Layer L2

Inputs Layer L2 is responsible for detailing the routing of robots in the waypoint network and for managing navigation conflicts. To do this, it first receives from L1 a high-level schedule specifying one decomposition method for each observation O_i and an associated start date. From this, it is possible to derive, for each robot $r \in \mathcal{R}ob$, the list of observations that must be successively realized by r, written as $obsSeq_r = [obsSeq_{r,1}, \ldots obsSeq_{r,|obsSeq_r|}]$ where $obsSeq_{r,i} \in \mathcal{O}bs$.

Layer L2 also takes as an input a representation of the navigation paths available in the waypoint graph, given by:

- a set of *links* $\mathcal{L} = [1..|\mathcal{L}|]$, where a link is a direct connection between two adjacent waypoints;
- for each robot r and each pair of successive observations (i, i') realized by r, a list of n candidate paths $P_{r,i,i'} = [P_{r,i,i',1}, \ldots, P_{r,i,i',n}]$ which can be used by r to go from i to i' (typically the n shortest paths); each path $P_{r,i,i',p}$ is specified by a sequence of m links $LinkSeq_{r,i,i',p} = [LinkSeq_{r,i,i',p,1}, \ldots, LinkSeq_{r,i,i',p,m}]$ successively traversed on that path, where $LinkSeq_{r,i,i',p,k} \in \mathcal{L}$.

Hierarchical scheduling problem The HSP built for layer L2 is illustrated in Fig. 3. It contains two kinds of disjunctive resources: the frequency resources, used to emit observation data as in L1, and the link resources, consumed by robot moves (two robots cannot simultaneously be on the same link). The root task network (T_0, Ψ_0) contains the sequence of high-level moves and observations realized by each robot (two robots in the example provided). In Fig. 3, the sequences of observations of robots are $obsSeq_1 = [2, 4]$ and $obsSeq_2 = [3, 1, 5]$. Robot 1 must therefore realize a compound move $MV_{1,0,2}$ from its initial location to observation number 2, then a primitive observation task O_2, then a move $MV_{1,2,4}$ from observation 2 to observation 4, and last a primitive observation task O_4. Each observation task consumes the frequency used by the robot. In the root task network, precedence constraints are imposed over these tasks to guarantee that each robot realizes an adequate move activity between two successive observations. The root task network also contains precedence constraints between some observations (coming from the precedence required between observation requests).

Each compound move $MV_{r,i,i'}$ between two observations i, i' has as many decomposition methods as the number of possible paths $P_{r,i,i',p}$ usable between the two observations (two possible paths in the case of compound move $MV_{2,3,1}$). A decomposition using path $P_{r,i,i',p}$ points to a task network which specifies a sequence of atomic moves $mv_{r,i,i',p,l}$ required on links of the waypoint graph. Each atomic move consumes the link resource number $LinkSeq_{r,i,i',p,l}$. For instance, the first path for $MV_{2,3,1}$ is the sequence of links $[l_3, l_9, l_8]$ and each subtask $mv_{2,3,1,1,k}$ consumes the kth link of the sequence. As for layer L1, a CP encoding can then be directly obtained from all these specifications.

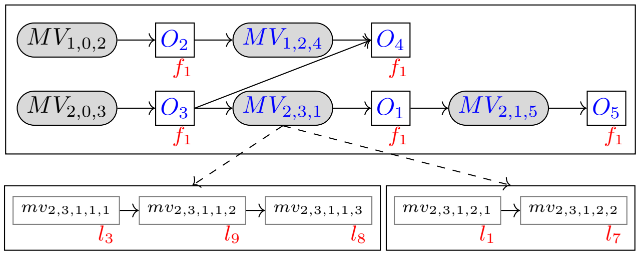

Figure 3: HSP for layer L2

3.3 Iterative Hierarchical Scheduling

We now present the interaction between the two layers modeled previously. As explained previously, we define an iterative process in which the solution found by the second layer is used to update the inputs used by the first one. An illustration is given in Fig. 4 for the robot observation mission: layer L1 transmits the sequence of observations for each robot to layer L2, and the solution produced by the latter is used to update the duration matrices for each robot. The iterations between layers are performed until a given CPU time is reached.

Note that the purpose of this process is not to obtain an optimal solution for the real problem but to get solutions of good quality within a short time. In fact, as layer L2 solutions are constrained by the solutions of layer L1 that works with an approximation of the global model, the iterative process has no guarantee to find an optimal solution.

Algorithm 1 presents a generic pseudo-code of a process between two layers that use each others solutions in order to minimize an objective function. We first detail the main

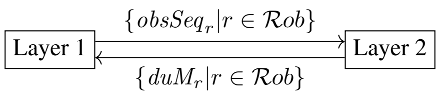
$\{obsSeq_r | r \in \mathcal{R}ob\}$
Layer 1
Layer 2
$\{duM_r | r \in \mathcal{R}ob\}$

Figure 4: Iterative process using layers.

lines of this pseudo-code and then detail our implementation for the multi-robot application.

The first step of the algorithm is to initialize several elements. The last solution found by the different layers sol_1 and sol_2 and the best solution found by layer L2 sol^* are all set to $null$. The best objective value found by L2 is denoted obj^* and is set up to an initial upper bound obj^{UP} (line 1). The input data for layer L1 are initialized through function **initL1** (line 2).

The process runs until a maximum CPU time is reached (line 3). In order to escape from local optima, we follow a restart-based strategy modeled by function **shouldRestart** that takes as parameters the last solutions found by the two layers. If a restart is required, function **perturbL1** re-initializes a given percentage of the input data of layer L1 (line 4). Then, if layer L2 has found a solution at the previous iteration, the latter is used by function **updateL1** to update the input data of the first layer (line 5). The problem to solve by L1 is created (line 6) and the associated solution is obtained through function **solve** to which a maximum CPU time is given. In our case, this CPU time is computed by a function **maxTime** that uses the number of iterations $nLoops$ desired for the whole process and the CPU time elapsed (line 7). More complex strategies could be defined to share the CPU time between L1 and L2. If a solution exists for layer L1, then it is used to create the problem pb_2 that is solved by layer L2 (lines 8-10). If this problem has a solution, we first compare its objective value to the best one found so far and update the latter if needed (lines 11 - 12). The best solution is finally returned (line 14).

Algorithm 1: **IterativeHS**($obj^{UP}, cpuMax, nLoops$):

```
1  $sol_1, sol_2, sol^* \leftarrow null$, $obj^* \leftarrow obj^{UP}$, $it \leftarrow 0$
2  initL1()
3  while CpuTimeElapsed() $< cpuMax$ do
4      if shouldRestart($sol_1, sol_2$) then perturbL1()
5      else if $sol_2 \neq null$ then updateL1($sol_2$)
6      $pb_1 \leftarrow$ createPbL1()
7      $sol_1 \leftarrow$ solve($pb_1$, maxTime($it, nLoops$))
8      if $sol_1 \neq null$ then
9          $pb_2 \leftarrow$ createPbL2($sol_1$)
10         $sol_2 \leftarrow$ solve($pb_2$, maxTime($it, nLoops$))
11         if $sol_2 \neq null \wedge$ objective($sol_2$) $< obj^*$ then
12             $obj^* \leftarrow$ objective($sol_2$), $sol^* \leftarrow sol_2$
13     $it \leftarrow it + 1$
14 return $sol^*$
```

We now detail how the functions and parameters defined previously are instantiated for the multi-robot application.

- **objective**, obj^{UP} The objective function is to minimize the makespan and H (i.e. the length of the temporal horizon) is used as an initial upper bound.
- **initL1** The data to be initialized for layer L1 are the approximate setup time matrices for each robot r. Depending on how the initialization is chosen, the algorithm can behave differently. If the matrices are initialized with lower bounds of the real duration (for instance, duM_r),

then layer L1 tends to provide optimistic solutions to layer L2 and the makespan of solutions of layer L2 can be greater than the ones of layer L1. On the contrary, if the matrices are initialized with upper bounds of the real duration, then layer L1 provides pessimistic solutions to layer L2. In our implementation, we have defined an optimistic layer L1 by using the matrices duM_r that give the shortest duration to go from one waypoint to another.

- $\mathbf{shouldRestart}$ As we have implemented an optimistic layer L1, we expect the makespan of the solutions it produces to be lower than the one of layer L2. Therefore, we restart whenever layer L1 produces a solution with a makespan bigger than the best one found so far. For layer L2, we allow solutions with lower quality in order for layer L1 to improve its approximation, but we restart if layer L2 has the same makespan during a given number of consecutive iterations.
- $\mathbf{perturbL1}$ We randomly reinitialize a percentage $rateReinit$ of the duration matrix of each robot.
- $\mathbf{updateL1}$ To update the duration matrices of layer L1 based on the solution given by layer L2 at the previous iteration, we have defined a reinforcement learning rate parameter $\alpha \in [0, 1]$ that represents the influence of duration of move tasks from solution of layer L2 (denoted $duML2_r$) on the input values of layer L1 (denoted $duML1_r$). Formally, $duML1_r$ is updated by $(1 - \alpha) \cdot duML1_r + \alpha \cdot duML2_r$.

The functions for creating and solving problems of each layer are implemented along with the description given previously in the paper.

4 Experiments

Instances Experiments were performed over several Multi-Robot problem instances generated randomly, where the field structure contains $3 \times |\mathcal{R}eq|$ waypoints connected to their closest neighbors within a fixed range. The generated instances contain from 1 to 15 *observation requests*, each request requiring observations from 1 to 3 robots. From 1 to 3 *frequencies* are available to transfer observation data, and 3 *robots* are available to carry out the observations, each of them with different specifications (particularly a different speed which determines the duration needed to traverse a link). Function $\mathbf{updateL1}$ is implemented with a learning rate parameter α ranging from 0.2 to 1 and the reinitialization rate is $rateReinit = 0.2$.

The generated instances were all tested on IBM ILOG CP Optimizer 12.5 on an Intel Xeon CPU E5-1603, 2.80GHz 8GB RAM, setting $cpuMax = \{5, 10, 30\}$ minutes and an adequate iterations number $nLoops$, depending on the number of *observation requests* and the $cpuMax$.

Interaction between layers L1 and L2 To illustrate the interaction between the two layers, makespan results over iterations of the proposed approach are presented in Fig. 5 for a problem instance containing 5 *observation requests* (each one must be observed by 2 different robots) and 3 available *robots*. In order to accentuate the behavior, we have specifically implemented a very optimistic $\mathbf{initL1}$ function with matrix values duM_r equal to zero and we have used a parameter $\alpha = 1$. The mean duration of one L1-L2 loop iteration is 1.188 seconds. The best obtained makespan is marked with a filled dot.

Fig. 5 shows that makespans of both layers tend to converge quickly. When solutions cannot be improved, restarts are realized (vertical lines in the figure). More precisely, the first two restarts occurred when layer L1 did not find a better solution than the best one found so far, and the third one because the makespan of the solution in layer L2 remained the same during several iterations. Since $\mathbf{initL1}$ is very optimistic, the makespan of solutions of layer L1 tends to increase, given that values of input duration matrices are also increasing.

Full Model To compare the hierarchical decision process with a global one-shot resolution strategy, we developed a full CP model of the robot deployment problem. This CP model contains (1) the model of layer L1 and (2) the model of layer L2 duplicated for each possible transition between observations. More precisely, the full CP problem model involves, for each robot r and each pair of distinct observations i, i', one optional interval $itv_{r,i,i'}$ representing a global move of robot r from i to i', plus a huge number of optional intervals $itv_{r,i,i',p,k}$ modeling the move of robot r on the kth link of the pth path available to transit from i to i'. To boost constraint propagation, the model contains, for each robot, both (1) the coarse-grain no overlap constraint of layer L1, which takes into account all observation intervals and minimum setup times between them, and (2) a fine-grain no overlap constraint that takes into account all observations and all move intervals over links associated with each robot. Last, a constraint is added to ensure that the successive fine-grain activities realized by each robot have consistent types, *i.e.* that an observation interval associated with observation i is preceded by a move interval of the form $itv_{r,i',i}$.

Comparison between the Two-Layer process and the Full Model All generated instances have been tested on the two-layer model and the full model with different maximum CPU times. Representative results are presented in Fig. 6 and Fig. 7. For the instances of Fig. 6, only one robot must observe each request, and there are $0.2 \times |\mathcal{R}eq^2|$ randomly generated precedences between requests. For Fig. 7, two different robots must observe each request and there are no precedences. Since the same makespan is reached for most of the experiments with several α values and the impact on the behavior is not significant, we only present results for $\alpha = 0.7$. Similarly, giving more CPU time to the two-layers process does not improve significantly the best solution and we only present the results for 5 minutes.

For the full model, the generated instances contain from 14 interval variables in the smallest instance (with only one *observation request*) to 139543 ones in the largest instance. For the largest instances, the full model does not find any solution, even with a CPU time of 30 minutes. The two-layer approach achieves better makespan results in a significantly shorter time. It provides first solutions of good quality in less than 3 seconds (not represented on the figures), even for the largest instances in which the complete solver is not able to reach any solution after several minutes. For the smallest in-

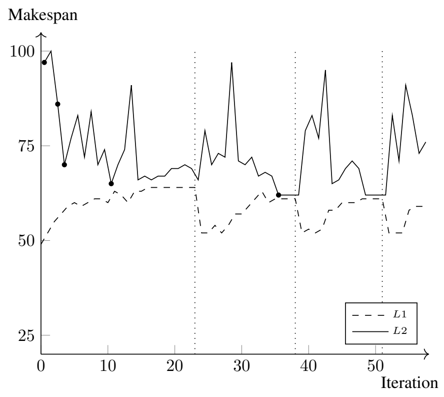

Figure 5: Interaction between layers

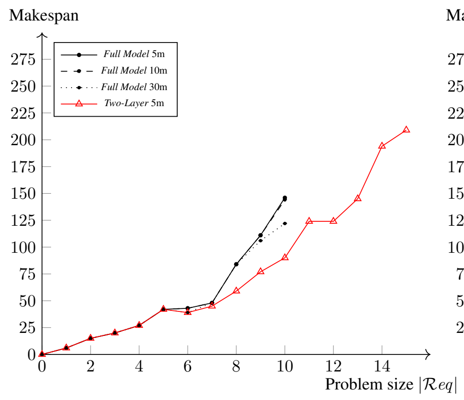

Figure 6: Makespan results with 1 robot per request and precedences

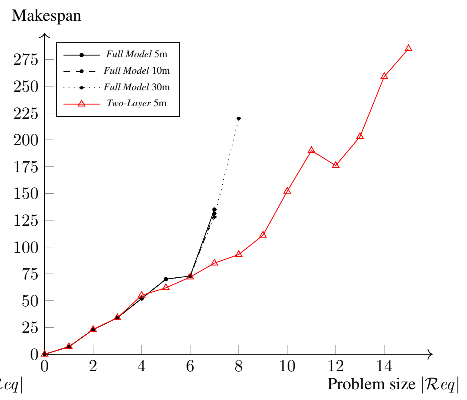

Figure 7: Makespan results with 2 robots per request and no precedences

stances, the two-layer approach manages to find the optimal solution (without proving its optimality).

These results demonstrate that the proposed iterative approach is more effective than the full model for solving large hierarchical scheduling problems. It allows to quickly get first solutions with good quality no matter the problem size, which represents the biggest drawback when it comes to solving with the full model.

5 Related Works

As said before, HSPs are related to HTNs (Erol, Hendler, and Nau 1994; Nau et al. 2005), in which high-level tasks are broken down into primitive tasks using a catalog of decomposition methods. One of the main difference is that in HTNs, tasks basically interact through their preconditions and effects on the system state, whereas in HSPs they interact through their consumptions on resources. In standard HTNs, it is still possible to consider state features representing the current amounts of resources available, however this can make quite complex the reasoning about resource usages at different levels of the decision hierarchy. Some recent HTN planners explicitly integrate resources and temporal constraints, such as GSCCB-SHOP2 (Qi et al. 2017). The latter is however not available for comparing results.

On the planning side again, several hierarchical planners can reason about global abstract plans and progressively decompose the abstract tasks of these plans. This is the case for planners such as HiPOP (Bechon et al. 2014), FAPE (Dvorak et al. 2014), and ASPEN (Chien et al. 1999). The CHIMP hierarchical planner (Stock et al. 2015) is even able to handle online action plans containing subtasks shared between high-level tasks. To our knowledge, these planners do not benefit in their current state from the strength of constraint-based scheduling for reasoning about resources.

On the algorithmic side, the two-layer scheduling approach obtained can first be related to work on approximation models (or *surrogate models*) for black-box optimization (Khac Vu et al. 2016). Indeed, for layer L1, the computation of a high-level solution through the combinatorial model of layer L2 can be seen as the computation of a complex evaluation function. The latter is summarized in layer L1 by a matrix of minimum transition durations between observations. Each time a new detailed schedule is available, this surrogate model is updated based on the real transition times that take into account interferences between atomic moves on links of the waypoint graph. In surrogate models, a key point is the choice of the next parameters for which the complex evaluation function (layer L2) must be computed. In our case, we evaluate at each step the more promising high-level schedules according to layer L1.

The interaction between layers can also be compared to Bender's decompositions (Benders 1962), where the solutions produced by a MIP-based first-stage problem are sent to a second-stage problem that might produce new constraints (Benders cuts) which are violated by the current solution of the first-stage problem. In our case, the differences are that decision layer L1 is based on a CP model and not on MIP, and layer L2 does not need to compute cuts. Instead, it just returns a fine-grain plan from which the parameters of the coarse-grain problem of layer L1 are updated.

Last, in constraint programming, representation hierarchies have already been used for configuration problems in which the specification of components has to be progressively refined (Mailharro 1998). One of the main differences is that HSPs cover hierarchies of tasks for scheduling problems, which leads to other solving techniques.

6 Conclusion

This paper introduces techniques for scheduling hierarchical tasks. As shown in the experiments, the approach exploits the strengths of already existing CP solvers and gives acceptable computation times, even on problems for which the set of possible task decompositions is large. On the modeling side, we could add representation features to deal with action preconditions and action effects as in HTNs, and benefit from recent work on the CP encoding of such specifications (Bit-Monnot 2018). On the algorithmic side, we could search for strategies that decompose compound tasks step-by-step, instead of having an arbitrary separation between two decision layers.

References

Bechon, P.; Barbier, M.; Infantes, G.; Lesire, C.; and Vidal, V. 2014. Hipop: Hierarchical partial-order planning. In *In Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS'14)*.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1):238–252.

Bit-Monnot, A. 2018. A constraint-based encoding for domain-independent temporal planning. In *International Conference on Principles and Practice of Constraint Programming*, 30–46. Springer.

Brucker, P.; Drexl, A.; Möring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research* 112(1):3–41.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999. Using iterative repair to increase the responsiveness of planning and scheduling for autonomous spacecraft. In *International Joint Conference on Artificial Intelligence (IJCAI 1999)*.

Dvorak, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and acting with temporal and hierarchical decomposition models. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, 115–121.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, 1123–1128. AAAI Press.

Khac Vu, K.; D'Ambrosio, C.; Hamadi, Y.; and Liberti, L. 2016. Surrogate-based methods for black-box optimization: Surrogate-based methods for black-box optimization. *International Transactions in Operational Research* 24:393–424.

Mailharro, D. 1998. A classification and constraint-based framework for configuration. *AI EDAM* 12:383–397.

Nau, D.; Au, T. C.; Ilghami, O.; Kuter, U.; Wu, D.; Yaman, F.; Munoz-Avila, H.; and Murdock, J. W. 2005. Applications of shop and shop2. *IEEE Intelligent Systems* 20(2):34–41.

Qi, C.; Wang, D.; Muñoz-Avila, H.; Zhao, P.; and Wang, H. 2017. Hierarchical task network planning with resources and temporal constraints. *Knowledge-Based Systems* 133:17–32.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6459–6464.