Modeling scheduling problems with Hexaly

Léa Blaise [*]

[*] *Hexaly, 251 Boulevard Pereire, 75017 Paris, France (e-mail: lblaise@hexaly.com).*

Abstract: In this paper, we focus on scheduling problems in general, and on how to model the constraints typically encountered in these problems using Hexaly's modeling formalism. Hexaly is a global mathematical solver offering a rich nonlinear and set-based modeling formalism. We show how we can use its collection of decision variables (lists, intervals, and optional intervals) and operators (set-based operators, variadic operators, lambda functions) can be used to efficiently model characteristics such as disjunctive and cumulative resources, preemption, or multi-alternatives. We also show that its compact models allow to solver to reach very good performance on scheduling problems.

Keywords: scheduling, modeling, solver

1. INTRODUCTION

Hexaly is a global optimization solver based on various exact and heuristic techniques described in Gardi et al. (2014). In this paper, we focus on its performance on scheduling problems in general (problems with precedence constraints and disjunctive or cumulative resources, minimizing makespan, weighted sum of tardiness, etc.). Hexaly's modeling formalism makes it easy to express many academic and industrial scheduling problems, using only generic operators. These models, based on interval and list decision variables, have the advantage of being compact, enabling the solver to handle large-scale problems. Indeed, the algorithms implemented within Hexaly exploit this modeling and enable it to achieve very good performance on various scheduling problems. As shown in Table 1, the solver reaches gaps to the best known solution from the literature below 2% with up to 2,000 tasks on various academic scheduling problems, all within one minute of running time.

Problem	max # tasks	Gap
Job Shop	2,000	2,2%
Flexible Job Shop	500	0.5%
Open Shop	400	0.1%
RCPSP	300	1.4%

Table 1. Gap to the best known solution from the literature on various scheduling problems

In this paper, we will show how the core features of Hexaly's modeling formalism, such as list and interval variables and lambda functions, enable users to write straightforward and efficient models for disjunctive and cumulative scheduling problems. We will also show that models written using Hexaly's set-based formalism are more compact than their equivalents in the linear or constraint programming formalisms used by other classical solvers, such as Gurobi or CP Optimizer.

2. HEXALY'S SET-BASED MODELING FORMALISM

In addition to classical Boolean, integer, and float decisions, Hexaly's modeling formalism offers set-based decision variables, such as intervals and lists. Relying on these decisions, we can write compact and efficient models for scheduling problems.

An interval is a decision variable whose value is an integer interval of the form $[a, b)$, verifying $a \leq b$. They typically represent time intervals. Indeed, they are used in scheduling models to represent the time span during which each task is being processed.

In disjunctive scheduling problems, the tasks are scheduled one after the other, and it is often useful to know their processing order. For that, we use list decision variables. A list of domain size n is a decision variable whose value is a permutation of a subset of $[0, n-1]$. In a scheduling problem, if each element between 0 and $n-1$ represents the index of a task, then the list variable represents the ordered set of the tasks that are scheduled.

Another specificity of Hexaly's modeling formalism is the possibility to iterate over a variable number of terms. Any n-ary operator (such as *sum*, *product*, *min*, *max*, *and*, *or*, and *xor*) can indeed be called with two arguments: a set of integers S, and a function f. The set of integers S does not have to be fixed, and its contents can vary during the search. For example, it can be a range $[0, x]$, where x is an integer decision variable, an interval decision variable, or a list decision variable.

3. MODELING DISJUNCTIVE SCHEDULING PROBLEMS

3.1 Modeling non-overlap constraints

The presence of disjunctive resources in a scheduling problem is characterized by non-overlap constraints on tasks assigned to the same resource. A simple way of writing this constraint consists in exploiting the order of the tasks: each task can only start after the previous task has finished.

With Hexaly's modeling formalism, this constraint is written using two types of decision variables. On the one hand, the resource is represented by a list variable: the elements of the list correspond to the tasks executed on the resource, arranged in ascending order of execution dates. In addition, interval decision variables are used to represent the execution range of each task.

```
order <- list(nbTasks);
task[0...nbTasks] <- interval(0, horizon);
```

We then write the non-overlap constraint using a variadic *and* operator and a lambda function. This constraint reads "for any position i in the list, the task at position i must be executed before the task at position $i+1$". This formulation based on list and interval variables has the advantage of allowing the user to write the non-overlap constraint in $O(n)$ only, where n represents the number of tasks, as opposed to $O(n^2)$ when writing pairwise non-overlap constraints on the interval variables.

```
constraint and(0...nbTasks-1, i => task[order[i]]
    ↪ < task[order[i+1]]);
```

3.2 Modeling flexible disjunctive resources

In a flexible disjunctive scheduling problem, several resources are available, and we must decide which resource will execute each task. In this case, if n is the number of tasks, we use one list variable of domain size n per resource, representing the indices of the tasks scheduled on this resource, in ascending order of execution dates. To ensure that each task is scheduled on exactly one resource, we constrain the lists to form a partition of $[0, n-1]$. We can then use a similar formulation as in the previous section to model the non-overlap constraints on the resources: for any given resource j, and for any position i between 0 and the number of tasks scheduled on resource j minus 2, the task at position i on machine j must be executed before the task at position $i+1$ on machine j.

```
order[j in 0...nbMachines] <- list(nbTasks);
constraint partition[j in 0...nbMachine](order[j]);
for [j in 0...nbMachines] {
    constraint and(0...count(order[j])-1, i =>
    ↪ task[order[i]] < task[order[i+1]]);
}
```

The size of the non-overlap constraint for resource j is then $O(n_j)$, where n_j is the number of tasks scheduled on resource j. Since the affectation of a task to a resource is a decision of the problem, the values of n_j for each resource j are not fixed but can vary during the search. Thanks to the variadic formulation, it is then possible to model the non-overlap constraints in such a way that the size of each individual constraint varies and the total size of all constraints remains $O(n)$, making the model very compact.

Using Hexaly's modeling formalism, a scheduling problem with n tasks and m disjunctive flexible resources then uses $O(n+m)$ decision variables: n intervals and m lists. We can compare this number with the number of decisions in equivalent models written using other modeling formalisms. For instance, a model written for a constraint programming solver like CP Optimizer uses $O(nm)$ interval decision variables: one for each task and each resource. As described in Laborie et al. (2018), to ensure that each task is scheduled on only one resource, and not on all resources at once, the m intervals associated with the same task are linked together by an alternative constraint, allowing only one of the intervals to be considered in the schedule. In a linear programming formalism, a classical formulation of the problem such as the one given by Özgüven et al. (2010) uses $O(n^2m)$ Boolean and integer decisions. Thanks to its compact modeling, Hexaly is able to handle much larger instances. The comparison of the number of decision variables for these three modeling formalisms is summarized in Table 2.

	Hexaly	CP Optimizer	MIP
Boolean	0	0	n^2m
Integer	0	0	n^2m
Interval	n	nm	0
List	m	0	0
Total	$n+m$	nm	n^2m

Table 2. Comparison of the number of decision variables for different modeling formalisms

Its compact modeling formalism and generic heuristic search, described in Blaise (2022), allow Hexaly to achieve very good performance on disjunctive scheduling problems. For instance, it reaches an average optimality gap of 0.5% in one minute of running time on the Flexible Job Shop Scheduling Problem (FJSP) research benchmark by Behnke and Geiger (2012), with up to 500 tasks and 60 machines. Moreover, it also allows Hexaly to deal with much larger instances. For example, on the very large-scale Job Shop Scheduling Problem (JSSP) benchmark proposed by Da Col and Teppan (2022) (1,000 jobs and 1,000 machines, for a total of 1,000,000 tasks), Hexaly improves the literature's best known solution by 8.6% on average within 10 minutes of running time.

4. MODELING CUMULATIVE SCHEDULING PROBLEMS

Unlike disjunctive resources, a cumulative resource can handle several tasks at once, within the limits of its capacity. Similarly as for disjunctive scheduling problems, the models for cumulative scheduling problems use interval decision variables representing the time span of the tasks. However, as the tasks assigned to a cumulative resource are not ordered, the models do not require the use of list decision variables.

The constraint associated with the presence of a cumulative resource can be expressed as follows. At any time t, the sum of the weights of the tasks running on the resource must be less than the resource's capacity. For any time t and any task i, we can check whether task i is running at time t by using the *contains* operator on the interval decision representing task i:

```
contains(task[i], t)
```

We can then compute the weighted sum of these quantities over all the tasks to check whether the resource capacity is respected at time t:

```
sum[i in 0...nbTasks] (weight[i] * contains(task[i], t))
    ↪ <= capacity
```

Finally, we must ensure that the constraint is verified at any time t. For that, we use a variadic *and* formulation and a lambda function to iterate over the entire time horizon:

```
constraint and(0...horizon, t => sum[i in 0...nbTasks]
    ↪ (weight[i] * contains(task[i], t)) <= capacity);
```

Thanks to the variadic operators available in Hexaly's modeling formalism, the formulation of the cumulative constraints is very compact. Indeed, the variadic *and* operator iterating over the time horizon is never "unrolled". On the contrary, both the user model and the internal representation of the constraint stay compact throughout the resolution.

These constraints form the foundation of many cumulative scheduling problems, such as the Resource-Constrained Project Scheduling Problem (RCPSP). Thanks to its efficient modeling and interval representation of cumulative constraints, Hexaly achieves very good performance on such problems. For example, the solver reaches an optimality gap of 1.9% within 1 minute of running time on the academic benchmark proposed by Debels and Vanhoucke (2007) for the Resource-Constrained Project Scheduling Problem (RCPSP), composed instances with 300 tasks.

5. MODELING OPTIONAL TASKS IN SCHEDULING PROBLEMS

Interval and list decision variables make the modeling of most academic and industrial scheduling problems very straightforward. However, scheduling problems with optional tasks remain hard to model. In this section, we show how introducing optional interval decision variables allows to model certain complex aspects of scheduling problems, such as preemption and multi-alternatives.

5.1 Optional interval decision variables and presence *operator*

An optional interval is a decision variable whose value is either an interval of the form $[a, b)$, verifying $a \leq b$, or *absent*. The *absent* value means that the task corresponding to this interval is not scheduled. To facilitate the handling of optional intervals, Hexaly's operator library also includes a *presence* operator, returning *true* if the interval is present and *false* if it is absent.

It is generally more advantageous to use optional interval variables rather than regular intervals coupled with Boolean decisions representing these intervals' presence. Indeed, information about the presence and execution dates of a task is then held within a single decision, giving the solver more information about the problem structure, and enabling it to apply relevant algorithms to solve it. In addition, it is possible to implement default behaviors for operators that apply to optional intervals, making it easier to write models. For example, the *hull* operator takes as input a set of intervals and returns the hull of all present intervals, or *absent* if none of these intervals is present. Similarly, the *intersection* operator returns *absent* if at least one of the intervals is absent.

5.2 Modeling multi-alternative scheduling problems

We consider a scheduling problem with multiple alternatives. In this problem, tasks are linked by precedence constraints. The successors of a given task constitute different alternatives, only one of which must be realized. Each of these alternatives comprises one or more subtasks. If an alternative is selected, then each of its subtasks must be completed. Using optional interval decisions and nonlinear operators such as *presence* and *hull*, the problem is expressed in a simple and compact way:

```
subtask[i in 0...n][k in 0...nbTasks[i]] <-
    ↪ optionalInterval(0, H);
alternative[i in 0...n] <- hull[k in 0...nbTasks[i]](
    ↪ subtask[i][k]);
for [i in 0...n] {
    constraint sum[s in successors[i]](presence(alternative
    ↪ [s])) == presence(alternative[i]);
    for [k in 0...nbTasks[i]] constraint presence(subtask[i
    ↪ ][k]) == presence(alternative[i]);
}
```

5.3 Modeling pseudo-preemptive scheduling problems

A preemptive scheduling problem is one in which tasks can be interrupted and then resumed. In a pseudo-preemptive version of the problem, a maximum number p of interruptions is defined for each task, which is then modeled as a set of $p + 1$ optional subtasks. At least one of these subtasks must be present, and the sum of durations of the present subtasks must be equal to the duration of the task. For ease of modeling, we also assume that if a subtask of index k is present, then all subtasks of index $j \leq k$ are also present. Here again, Hexaly's optional interval decision variables and numerous nonlinear operators (*presence*, *length*, ternary operator...) make it easy to model such problems:

```
subtask[i in 0...n][k in 0...p+1] <- optionalInterval(0, H)
    ↪ ;
for [i in 0...n] {
    constraint presence(subtask[i][0]);
    for [k in 1...p+1] constraint presence(subtask[i][k])
    ↪ <= presence(subtask[i][k-1]);
    constraint sum[k in 0...p+1](presence(subtask[i][k])?
    ↪ length(subtask[i][k]): 0) == d[i];
}
```

6. CONCLUSION

With Hexaly's modeling formalism, which includes list and interval decision variables, as well as variadic operators and lambda functions, writing models for academic and industrial scheduling problems is made very straightforward. Its collection of generic nonlinear and set-based operators can be used to model the typical characteristics of scheduling problems, such as disjunctive and cumulative resources, but also complex business constraints. These compact models are then made into efficient internal representations of the problem, which Hexaly exploits to deliver state-of-the art results on scheduling problems, including on very large-scale instances with millions of tasks.

REFERENCES

Behnke, D. and Geiger, M. (2012). Test instances for the flexible job shop scheduling problem with work centers. Technical report, BWL, insb. Logistik-Management.

Blaise, L. (2022). *Modélisation et résolution de problèmes d'ordonnancement au sein du solveur d'optimisation mathématique LocalSolver*. Theses, INSA de Toulouse.

Da Col, G. and Teppan, E.C. (2022). Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9, 100249. doi: https://doi.org/10.1016/j.orp.2022.100249.

Debels, D. and Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3), 457–469. doi:10.1287/opre.1060.0358.

Gardi, F., Benoist, T., Darlay, J., Estellon, B., and Megel, R. (2014). *Mathematical Programming Solver Based on Local Search*. John Wiley & Sons, Ltd.

Laborie, P., Rogerie, J., Shaw, P., and Vilim, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23, 210–250.

Özgüven, C., Özbakır, L., and Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*. doi: https://doi.org/10.1016/j.apm.2009.09.002.