Column elimination for scheduling problems

Vianney Coppé
Hexaly
e-mail: `vcoppe@hexaly.com`

Column elimination [2] is a recent framework for solving combinatorial optimization problems in which the solution consists of multiple sequences. It uses a *relaxed decision diagram* to compactly encode a superset of all feasible sequences. By solving a constrained network flow problem in this decision diagram (DD), a (relaxed) minimum-cost set of sequences can be found. If it happens to correspond to an exact solution, it is also guaranteed to be the optimal one. Otherwise, this relaxed solution gives a lower bound on the optimal cost, and the DD can be refined to *eliminate* the relaxed solution at hand before solving a new iteration of the flow problem. This technique was successfully applied to several vehicle routing and graph coloring problems [2, 3]. In this paper, we investigate how to apply it to disjunctive scheduling problems.

There are two main obstacles to applying column elimination to scheduling problems. First, while routing problems typically involve uniform vehicles or few vehicle classes, the machines considered in scheduling problems are usually all different. This is due to heterogeneous machine-task compatibilities and possibly machine-dependent task durations, setup and changeover times. We can deal with this variety of machines by creating a separate relaxed DD for each of them and embedding those into a single *meta*-DD. This meta-DD simply connects its root node to each machine-specific root node, and each machine-specific terminal node to its own terminal node. Since machine-task compatibilities are generally sparse, the size of the meta-DD remains reasonable.

The second obstacle lies in the presence of precedence constraints between tasks. In routing problems, it is always preferable to visit clients as early as possible. However, scheduling problems with precedence constraints may require machines to wait for a task to be completed on another machine before starting a given task. We first explain how to allow such idle times and then how to enforce precedence constraints. To model idle times, the DD must include transitions for a whole range of start times for each task. Clearly, representing all possible time-indexed transitions would yield very large DDs, especially when the horizon is loose. We propose to use the modeling introduced in [1] for single-machine constraint propagation in constraint programming, in which time information is not explicitly encoded in the DD but rather derived a posteriori. It works by performing a top-down and bottom-up pass on the DD to compute information such as the earliest and latest completion times for each node n, respectively denoted $ect(n)$ and $lct(n)$. This information is used to identify and filter infeasible transitions, and to assign transition costs to the remaining arcs.

Because they are computed with respect to the most favorable completion time, the transition costs may be relaxed. Therefore, after finding a minimum-cost flow in the DD, we compute the true completion time of each task in the solution and use it to obtain the true transition costs. If all the transition costs match, the solution is exact. Otherwise, we identify transitions having a relaxed cost and refine them as follows. Let $a = (u, v)$ be an arc with relaxed cost and let ct_a be the true completion time of the corresponding transition in the solution. We refine the DD by splitting node v into two nodes v_1 and v_2 with shorter completion time intervals. More precisely, if $ect(v) < ct_a \leq lct(v)$, we create nodes v_1 and v_2 with explicitly encoded completion time intervals $[ect(v_1), lct(v_1)] = [ect(v), ct_a - 1]$ and $[ect(v_2), lct(v_2)] = [ct_a, lct(v)]$. We call this operation *splitting* because all the incoming and outgoing arcs of v are then transferred to nodes v_1 and v_2 before removing node v. After a certain number of such refinements, a new round of propagation and filtering can be performed to filter additional infeasible transitions and improve the arc costs.

Let us now explain how to enforce precedence constraints. We propose to handle them by embedding column elimination in a *branch-and-bound* (B&B) framework. After applying column elimination in a given B&B node, we look for violated precedence constraints in the solution. If there are none, the solution is feasible, and may later be proven optimal by the B&B. Otherwise, let i and j be two tasks with start and completion times st_i, ct_i and st_j, ct_j in the solution, such that i must precede j but $ct_i > st_j$. We create a first branch with the additional constraint $st(j) < ct_i$ and a second with $st(j) \geq ct_i$. Those time constraints can be propagated through the precedence network to tighten the earliest and latest times for each task. They are then injected into column elimination by finding all transitions that are incompatible with them and blocking any flow through those. Note that branching information can also be used to strengthen the arc costs and to determine the true completion times inside the refinement procedure.

It may happen that a combination of branching constraints creates an infeasible subproblem. To detect those cases, we apply column elimination on the linear relaxation of the constrained network flow problem and solve it with a linear programming solver. Therefore, we may also need to branch to separate fractional solutions – a procedure called *branch-and-refine* in [2] – using task time constraints and forced/forbidden machine-task assignment constraints.

This approach was implemented inside the Hexaly global optimization solver and significantly improves the bounds computed for several variants of the (flexible) job shop scheduling problem, increasing the proportion optimality proofs on the benchmark instances. It can handle all standard scheduling objectives; those including a *maximum* function need to use a slightly adapted flow problem.

References

[1] Andre A. Cire and Willem-Jan van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

[2] Anthony Karahalios and Willem-Jan van Hoeve. Column elimination for capacitated vehicle routing problems. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 35–51. Springer, 2023.

[3] Willem-Jan van Hoeve. Graph coloring with decision diagrams. *Mathematical Programming*, 192(1):631–674, 2022.