Chapter 4

Constraint Programming Formulations and Propagation Algorithms

This chapter presents the classical representation of RCPSP in constraint-based scheduling as well as the propagation algorithms for temporal constraints and resource capacity constraints (time-tabling, disjunctive reasoning, edge-finding, energetic reasoning, precedence energy and balance) used to solve the RCPSP through constraint-based scheduling.

4.1. Constraint formulations

4.1.1. *Constraint programming*

Constraint programming (CP) is based on a declarative description of a problem as (1) a set of decision variables with domains, i.e., sets of possible values, for instance the start time of an activity with its possible time-windows; and (2) a set of constraints restricting the combinations of values the decision variables can take, for instance a precedence constraint between the end time of activity A and the start time of activity B.

Constraints can be defined either explicitly as a set of compatible tuples of variable assignments or implicitly, for instance through an algebraic relation (e.g. $x \neq y$). A solution to a CP problem is a complete assignment of the variables satisfying all the constraints.

Chapter written by Philippe LABORIE, Wim NUIJTEN.

The search for a solution usually explores a search tree whose branches correspond to decisions (for instance instantiating a variable x to a given value v). During the search, constraints are used actively to remove inconsistent values from the current domain of variables. This process is known as *constraint propagation* [BES 06]. One way of propagating constraints consists of enforcing arc-consistency. A constraint is said to be arc-consistent if the domains of its variables have been reduced so that they only contain values that are supported by at least one solution to the constraint. For instance if the domain of variables x and y in constraint $C = (x \leq y)$ are so that $x \in [5, 10]$ and $y \in [0, 7]$, enforcing arc-consistency on C will reduce these domains to $x \in [5, 7]$ and $y \in [5, 7]$. At each node explored by the search, constraints are propagated until the fix point of the propagation is reached.

A particularly important class of constraints are the *global constraints*. Such constraints represent a set of basic constraints (for instance $\texttt{alldifferent}(x_1, ..., x_n)$ standing for $\forall i \in [1, n], j \in [i + 1, n], x_i \neq x_j$). Algorithms for propagating global constraints are able to propagate from a global point of view in an efficient way. The typical scheduling example of such a global constraint is the constraint that propagates on the combination of all activities requiring capacity from a non-renewable resource.

4.1.2. *Constraint-based scheduling*

Constraint-based scheduling is the discipline that studies how to solve scheduling problems by using CP. Over the years, it has grown into one of the most successful application areas of CP. Constraint-based scheduling extends CP by providing constructs such as activities and resources which have a scheduling semantics. An activity usually corresponds to at least 3 decision variables: start time, end time and processing time or duration. The domains of these variables represent the possible time-windows and duration of the activity. Ususal constraints in constraint-based scheduling are temporal constraints between activities and resource capacity constraints. Different types of resources can be modeled [BAP 06]: unary resources, cumulative resources, producible/consumable resources, state resources, etc. Resource demands represent a certain demand of resource (this quantity can be a decision variable) for a given activity. Although most of all the extensions described in Part II of this book can be modeled and solved using the paradigm of constraint-based scheduling, we focus in this chapter on the basic formulation of RCPSP and the underlying constraint propagation algorithms. More detailed surveys of constraint propagation algorithms for constraint-based scheduling are available in [DOR 99, BAP 01].

The basic RCPSP can be formulated, as shown in Table 4.1.2, in a constraint-based scheduling approach using OPL, the Optimization Programming Language [VAN 99]. The precedence graph structure is defined in line `8` as a set of precedence arcs `i,j`. The set of activities is defined in line `11`, activity `A[i]` being constructed with duration `p[i]`. An additional activity of duration `0` representing the makespan is defined in

line 12. The set of cumulative resources is created in line 13, resource R[k] being of capacity B[k]. Discrete resources post a global capacity constraint on the set of activities that require the resource. The objective function (minimize end time of makespan activity) is specified in line 14 followed by the constraints of the problem. Each activity A[i] is constrained to finish before the makespan activity (line 17) and to require b[i][k] units of resource R[k] in line 19. Finally, the precedence constraints of the precedence graph G are posted in line 22.

```
1  // Data
2  struct Precedence { int i; int j; };
3  int q            = ...; // Number of resources
4  int n            = ...; // Number of activities
5  int p[1..n]      = ...; // Processing times
6  int B[1..q]      = ...; // Resource capacities
7  int b[1..n,1..q] = ...; // Resource requirements
8  {Precedence} G   = ...; // Precedence graph
9
10 // Model
11 Activity A[i in 1..n](p[i]);
12 Activity makespan(0);
13 DiscreteResource R[k in 1..q](B[k]);
14 minimize makespan.end
15 subject to {
16   forall(i in 1..n) {
17     A[i] precedes makespan;
18     forall(k in 1..q : 0 < b[i][k])
19       A[i] requires(b[i][k]) R[k];
20   };
21   forall(<i,j> in G)
22     A[i] precedes A[j];
23 };
```

Table 4.1. *An OPL model of the basic RCPSP*

The next section describes the principles of the classical algorithms for propagating temporal constraints (section 4.2.1) and resource constraints (sections 4.2.2-4.2.8).

4.2. Constraint propagation algorithms

4.2.1. *Temporal constraints*

Temporal relations between activities can be expressed by linear constraints between start and end variables of activities. For instance, a standard precedence constraint between two activities A_i and A_j stating that A_j is to be started after A_i has ended can be modeled by the linear constraint $C_i \leq S_j$. In the general case of RCPSP with time lags, with both x and y a start or end variable and δ an integer, temporal relations can be expressed by constraints of the type $x - y \leq \delta$.

When the temporal constraint network is sparse, as it is usually the case in RCPSP, such constraints can be easily propagated using a standard arc-consistency algorithm [LHO 93]. In addition, a variant of Ford's algorithm (see for instance [GON 84]) proposed by Cesta and Oddi [CES 96] can be used to detect any inconsistency between such constraints in time polynomial to the number of constraints and independent of the domain sizes.

When the temporal network is dense or when it is useful to compute and maintain the minimal and maximal delay between any pair of time points in the schedule, path consistency can be enforced on the network [DEC 91] for example by applying Floyd-Warshall's All-Pairs-Longest-Path algorithm [FLO 62].

4.2.2. *Timetabling*

Timetabling relies on the computation of the minimal resource usage by the current activities in the schedule at every date t [Le 94b]. This aggregated demand profile is maintained during the search. It allows restricting the domains of the start and end times of activities by removing the dates that would lead to an over-consumption of the resource.

Suppose that activity A_i requires $b_{ik} \in [b_{ik}^-, b_{ik}^+]$ units of resource R_k and is such that $LS_i < EC_i$. Then we know that A_i will surely execute over the time interval $[LS_i, EC_i)$ requiring at least b_{ik}^- units of R_k; see activity A_1 in Figure 4.1 for an illustration. The minimal contribution of activity A_i at time t is thus given by the function:

$$U_k(A_i, t) \quad = \quad \begin{cases} b_{ik}^- & \text{if } LS_i \leq t < EC_i \\ 0 & \text{otherwise} \end{cases}$$

For each resource R_k, a curve $U_k(t)$ is maintained that aggregates all these demands: $U_k(t) = \sum_{i=1}^{n} U_k(A_i, t)$.

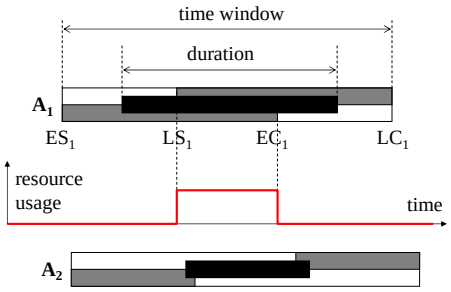

Figure 4.1. *Timetabling*

It is clear that if there exists a date t such that $U_k(t)$ is strictly greater than the maximal capacity of the resource B_k, the current schedule cannot lead to a solution and the search must backtrack .

Furthermore, $U_k(t) - U_k(A_i, t)$ represents the requirement by all activities except A_i on R_k at date t. If there exists a date t such that $U_k(t) - U_k(A_i, t) + b_{ik}^- > B_k$, then activity A_i cannot be executing at time t. In particular, if the above relation is true for some t with $ES_i \leq t < EC_i$ (resp. $LS_i \leq t < LC_i$), it allows to increase (resp. decrease) the earliest start time (resp. latest completion time) of activity A_i until the first date t' such that $U_k(t') - U_k(A_i, t') + b_{ik}^- \leq B_k$. Similar reasoning can be applied to find new upper bounds on the quantity of resource required by activities. The main advantage of the timetabling technique is its low practical algorithmic complexity, which is linear or even sub-linear in the number of activities. For this reason, it is the main technique used today for scheduling renewable resources on large problems.

4.2.3. *Disjunctive reasoning*

Let A_i and A_j be two activities on a resource R_k such that $b_{ik}^- + b_{jk}^- > B_k$. As such, they cannot overlap in time and thus either A_i precedes A_j or A_j precedes A_i, *i.e.*, the disjunctive constraint holds between these activities. In general the disjunctive constraint achieves arc-consistency on the formula:

$$[b_{ik} + b_{jk} \leq B_k] \vee [C_i \leq S_j] \vee [C_j \leq S_i]$$

The disjunctive constraint can be propagated on all activities with a worst case complexity in $O(n \log n)$ [VIL 04].

4.2.4. *Edge-finding*

Let $\phi \subset \mathcal{A}$ be a subset of activities that require a given resource R_k of capacity B_k and $A_i \in \mathcal{A} \setminus \phi$. Let us denote:

– $ES_\phi = \min_{A_j \in \phi} ES_j$, the earliest start time of all activities in ϕ;

– $EC_\phi = \min_{A_j \in \phi} EC_j$, the earliest completion time of all activities in ϕ;

– $LC_\phi = \max_{A_j \in \phi} LC_j$, the latest completion time of all activities in ϕ;

– $W_\phi = \sum_{A_j \in \phi} p_j \cdot b_{jk}$, the global energy required by ϕ.

The basic idea of edge-finding and activity interval techniques is to ensure that for any subset of activities ϕ, resource R_k provides enough energy over the time interval $[ES_\phi, LC_\phi)$ to allow the execution of all the activities of ϕ, that is: $W_\phi \leq B_k \cdot (LC_\phi - ES_\phi)$. Constraint propagation is usually performed by applying the three following deduction rules:

1) If $B_k \cdot (LC_\phi - ES_{\phi \cup \{i\}}) < W_{\phi \cup \{A_i\}}$, then A_i must finish after all activities in ϕ, in particular $EC_i := max(EC_i, EC_\phi)$.

2) If $ES_\phi < ES_i < EC_\phi$ and $B_k \cdot (LC_\phi - ES_\phi) < W_\phi + b_{ik} \cdot (min(LC_\phi, EC_i) - ES_i)$, then at least one activity A_j in ϕ must precede A_i, otherwise A_i would start between ES_ϕ and EC_ϕ and there would not be enough energy to execute $\phi \cup \{A_i\}$ between ES_ϕ and LC_ϕ, thus $ES_i := max(ES_i, EC_\phi)$.

3) If $ES_i < ES_\phi < EC_i$ and $B_k \cdot (LC_\phi - ES_\phi) < W_\phi + b_{ik} \cdot (EC_i - ES_\phi)$, then A_i must finish after all activities in ϕ, in particular $EC_i := max(EC_i, EC_\phi)$.

These rules make it possible to update the earliest start or completion times of activities and symmetrical rules allow to update the latest start and completion times. Edge-finding ([NUI 94]) and activity intervals propagation ([CAS 96]) are very similar techniques that mainly differ in the way the propagation rules are triggered: edge-finding algorithms are global algorithms that perform all updates on a given resource whereas activity interval approaches are performed incrementally as soon as the time bound of a activity changes.

Note that specific edge-finding algorithms have been designed for the special case of disjunctive resources ($B_k = 1$) [BAP 01]. These algorithms can be implemented with a time complexity in $O(n \log n)$ [VIL 04].

4.2.5. *Energy reasoning*

Energy-based constraint propagation algorithms compare the amount of energy provided by a resource over some interval $[t_1, t_2)$ to the amount of energy required by activities that have to be processed over this interval. [ERS 91] proposes the following definition of the required energy consumption that takes into account the fact that activities cannot be interrupted. Given a resource R_k, an activity A_i and a time interval $[t_1, t_2), W_k(A_i, t_1, t_2)$, the *left-shift / right-shift* required energy consumption of resource R_k by activity A_i over $[t_1, t_2)$ is b_{ik} times the minimum of the three following durations.

– the length of the interval: $t_2 - t_1$;

– the number of time units during which A_i executes after time t_1 if A_i is left-shifted, *i.e.*, scheduled as soon as possible: $p_i^L(t_1) = \max(0, p_i - \max(0, t_1 - ES_i))$;

– the number of time units during which A_i executes before time t_2 if A_i is right-shifted, *i.e.*, scheduled as late as possible: $p_i^R(t_2) = \max(0, p_i - \max(0, LC_i - t_2))$.

This leads to $W_k(A_i, t_1, t_2) = b_{ik} \cdot \min(t_2 - t_1, p_i^L(t_1), p_i^R(t_2))$.

The left-shift / right-shift overall required energy consumption $W_k(t_1, t_2)$ over an interval $[t_1, t_2)$ is then defined as the sum over all activities A_i of $W_k(A_i, t_1, t_2)$.

It is obvious that if there is a feasible schedule, then $B_k \cdot (t_2 - t_1) - W_k(t_1, t_2) \geq 0$ for all t_1 and t_2 such that $t_2 \geq t_1$.

It is shown in [BAP 01] how the values of W_k can be used to adjust activity time bounds. Given an activity A_i and a time interval $[t_1, t_2)$ with $t_2 < LC_i$, it is examined whether A_i can end before t_2. If there is a time interval $[t_1, t_2)$ such that

$$W_k(t_1, t_2) - W_k(A_i, t_1, t_2) + b_{ik} \cdot p_i^L(t_1) > B_k \cdot (t_2 - t_1),$$

then a valid lower bound of the end time of A_i is

$$t_2 + \frac{1}{b_{ik}}(W_k(t_1, t_2) - W_k(A_i, t_1, t_2) + b_{ik} \cdot p_i^L(t_1) - B_k \cdot (t_2 - t_1)).$$

Similarly, when

$$W_k(t_1, t_2) - W_k(A_i, t_1, t_2) + b_{ik} \cdot \min(t_2 - t_1, p_i^L(t_1)) > B_k \cdot (t_2 - t_1),$$

A_i cannot start before t_1 and a valid lower bound of the start time of A_i is

$$t_2 - \frac{1}{b_{ik}}(C(t_2 - t_1) - W_k(t_1, t_2) + W_k(A_i, t_1, t_2)).$$

[BAP 01] presents a $O(n^3)$ algorithm to compute these time bound adjustments for all n activities. It is first shown there are $O(n^2)$ intervals $[t_1, t_2)$ of interest. Given an interval and an activity, the adjustment procedure runs in $O(1)$. As such, the obvious overall complexity of the algorithm is thus $O(n^3)$. An interesting open question is whether there is a quadratic algorithm to compute all the adjustments on the $O(n^2)$ intervals under consideration. Another open question at this point is whether the characterization of the $O(n^2)$ time intervals in [BAP 01] can be sharpened in order to eliminate some intervals and reduce the practical complexity of the corresponding algorithm. Finally, it seems reasonable to think that the time bound adjustments could be sharpened. Even though the energy tests can be limited (without any loss) to a given set of intervals, it could be that the corresponding adjustment rules cannot.

4.2.6. *Precedence graph*

The above constraint propagation algorithms reason on the absolute position of activities in time (through their time bounds ES_i, EC_i, LS_i, LC_i). They provide an efficient propagation when these time bounds are tight but the propagation becomes weaker when time-window of activities is large.

Some additional propagation algorithms have been designed that reason on the relative position of activities on resources (precedence relations in a precedence graph) rather than their absolute position only. As a consequence, these algorithms allow a much stronger propagation when the time-windows of activities are large and when the current schedule contains many precedence relations as it is usually the case in RCPSP. These algorithms require that a temporal network representing the relations between the time-points of all activities (start and end) using the point algebra of ([VIL 86]) is maintained during the search. We denote $\{\emptyset, \prec, \preceq, =, \succ, \succeq, \neq, ?\}$ the set of qualitative relations between the time points of the schedule S_i, C_i. The temporal network is in charge of maintaining the transitive closure of those relations. Let r be one of the above relations, we denote $\neg(x\ r\ y)$ if and only if the relation $x\ r\ y$ cannot be deduced from the network.

The initial set of relations consists of the precedences $S_i \prec C_i$ for each activity A_i and $C_i \preceq S_j$ for each precedence constraint of the problem. During the search additional precedence relation can be added as decisions or as the result of constraint propagation.

Figure 4.2 illustrates an example of such a precedence graph. An arrow between two time-points x and y means $x \prec y$. Once the transitive closure of the graph has been computed, the following relations hold: $S_3 \prec C_2$, $\neg(S_1 \prec C_4)$.

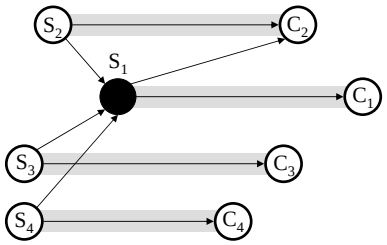

Figure 4.2. *Precedence graph*

4.2.7. *Energy precedence*

The *energy precedence* propagation algorithm ([LAB 03a]) for an activity A_i on a resource R_k ensures that for each subset Ω of predecessor activities of activity A_i the resource provides enough energy to execute all activities in Ω between ES_Ω and S_i. More formally, if $ES_\Omega = \min_{A_j \in \Omega} ES_j$ and $W_{\Omega,k} = \sum_{A_j \in \Omega} p_j \cdot b_{ik}$, it performs the following deduction rule:

$$\forall \Omega \subset \{A_j \in \mathcal{A}, C_j \preceq S_i\}, ES_i := max(ES_i, ES_\Omega + \lceil W_{\Omega,k}/B_k \rceil)$$

The propagation of the energy precedence constraint can be performed for all the activities A_i on a resource and for all the subsets Ω with a total worst-case time complexity of $O(n(p + log(n))$, where n is the number of activities on the resource and p the maximal number of predecessors of a given activity in the temporal network ($p < n$).

4.2.8. *Balance constraint*

On a renewable resource, the *balance constraint* ([LAB 03a]) can be defined as follows. The basic idea of the algorithm is to compute, for each activity A_i on a resource R_k, a lower bound on the resource usage at the start time of A_i (a symmetrical reasoning can be applied to perform some propagation based on a lower bound on the resource usage at the completion time of A_i). Using the temporal network a lower bound on the resource utilization at date $S_i + \epsilon$ just after the start time of A_i can be computed assuming that all the resource requirements that do not necessarily overlap S_i will not overlap it:

$$L_k(i) = \sum_{j/(S_j \preceq S_i) \wedge (C_j \succ S_i)} b_{jk}$$

Given this bound, the balance constraint is able to discover three types of information:

Dead ends. Whenever $L_k(i) > B_k$, the resource will surely be over-consumed just after time-point S_i so the search has reached a dead end.

New bounds on time variables. If $L_K(i) \leq B_k$, $\Delta(i) = B_k - L_K(i)$ represents a slack of capacity that must not be exceeded by all the resource requirements that, currently, do not necessarily overlap S_i but could overlap it. Let $P(i) = \{A_j / (S_j \preceq S_i) \wedge \neg(C_j \succ S_i)\}$. We suppose the activities $(A_{j_1}, \cdots, A_{j_u}, \cdots, A_{j_p})$ in $P(i)$ are ordered by decreasing earliest completion time EC_j. Let v be the index in $[1, p]$ such that:

$$\sum_{u=1}^{v-1} b_{j_u k} \leq \Delta(x) < \sum_{u=1}^{v} b_{j_u k}$$

If event S_i is executed at a date $S_i < EC_{j_v}$, not enough activities of $P(i)$ will be able to be completed strictly before S_i in order to ensure the resource is not over-consumed just after S_i as in this case, the consumed quantity will be at least $L_K(i) + \sum_{u=1}^{v} b_{j_u k} > B_k$. Thus, EC_{j_v} is a valid lower bound of S_i.

New precedence relations. Suppose there exists activity A_y in $P(i)$ such that:

$$\sum_{A_z \in P(i), C_z \succeq C_y} b_{zk} > \Delta(x)$$

Then, if we had $S_i \prec C_y$, we would see that again there is no way to avoid a resource over-consumption as it would consume at least:

$$L_k(i) + \sum_{A_z \in P(i), C_z \succeq C_y} b_{zk} > B_k$$

Thus, the necessary precedence relation: $C_y \preceq S_i$ can be deduced and added to the current temporal network.

On the precedence graph of Figure 4.2, if all activities require 1 unit of the same resource R_k of capacity 3, the balance constraint applied at time-point S_1 would compute $L_k(1) = b_{1k} + b_{2k} = 2$ and deduce that $\min(EC_3, EC_4) \leq S_1$.

The balance algorithm can be executed for all activities A_i with a global worst-case complexity in $O(n^2)$ if the propagation that discovers new precedence relations is not turned on and in $O(n^3)$ for a full propagation. In practice, there are many ways to shortcut this worst case and in particular, it was noticed that the algorithmic cost of the extra-propagation that discovers new precedence relations was in general negligible.

4.3. Conclusion

Constraint-based scheduling provides a wide spectrum of propagation algorithms for renewable resources ranging from sub-linear to quadratic time complexities. The

selection of a combination of propagation algorithms clearly depends on the size of the problem being solved.

For small problems until a few tens of renewable resources and a few hundred activities on each resource, a strong propagation scheme may be used. This is for instance the case of the MCS method described in section 7.6 and experimented on small RCPSPs. It uses a combination of timetabling, disjunctive, edge-finding, energy precedence and balance constraints.

Larger problems are generally solved using meta-heuristics. These methods, for instance the LNS described in section 7.6, are usually less sensitive to constraint propagation than tree search methods and light propagation schemes can be used, typically, only using timetabling.