Case-Based Project Scheduling

Mario Gómez* and Enric Plaza

IIIA - Artificial Intelligence Research Institute, CSIC - Spanish National Research Council, Campus U.A.B. 08193 Bellaterra. Catalonia (Spain)

Abstract. This paper presents a new approach for solving the Resource-Constrained Project Scheduling Problem using Case-Based Reasoning in a constructive way. Given a project to be scheduled our method retrieves similar projects scheduled in the past, selects the most similar project, and reuses as much as possible from the old solution to build a schedule for the project at hand. The result of this process is a partial schedule that is later extended and revised to produce a complete and valid schedule by a modified version of the Serial Schedule Generation Scheme. We present experimental results showing that our approach works well under reasonable assumptions. Finally, we describe several ways to modify our algorithm in the future so as to obtain even better results.

1 Introduction

Scheduling is the process of deciding how to commit limited resources among a variety of activities, subject to a set of constraints. Different scheduling domains pose different types of constraints. Examples of scheduling constraints include deadlines (e.g. activity i must be completed by time t), resource capacities (e.g. there are only four drills), and precedence constraints on the order of tasks (e.g. a piece must be sanded before it is painted). Scheduling is a class of optimization problems in which one seeks to minimize or maximize a function (e.g. minimizing the time span) while meeting some constraints. The Resource Constrained Project Scheduling Problem (RCPSP) is a class of scheduling problems consisting of a network of activities, precedence relationships and resource constraints (for general surveys on the subject see for example [1,2].

[3] showed that the RCPSP, as a generalization of the classical *job-shop scheduling* problem, belongs to the class of NP-hard optimization problems. Due to this complexity, proven optimal solutions have been computed only for very small problems; in other words, exact scheduling methods are mainly for generating benchmark solutions, and not for real-world applications. Approximate methods and heuristics are thus required for solving large scheduling problems as they usually appear in the real-world.

Synthetic tasks like planning and scheduling can be approached either by *constructive methods* or by *repair-based methods*. Constructive methods start with a description of requirements to build a completely new solution, while repair-based methods take a previous solution and some new requirements and modify

* Current affiliation is Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera s/n., 46022 Valencia, Spain.

B. Díaz Agudo and I. Watson (Eds.): ICCBR 2012, LNCS 7466, pp. 122–136, 2012.

the original solution to meet the new requirements. In this paper, we present a Case-Based Reasoning (CBR) framework for solving the Resource-Constrained Project Scheduling Problem (RCPSP) in a constructive way. This framework follows the general cycle of the CBR process: It begins with a retrieval step, which obtains past scheduling problems that are similar to the current problem; then, part of the solution to the most similar problem is reused and adapted to build a partial solution to the new problem; finally, this partial solution is extended and revised to produce a complete and valid solution by applying a modified version of the Serial Schedule Generation Scheme (SSGS).

The paper is organized as follows: Section 2 describes the RCPSP and the SSGS; Section 3 describes our framework for solving the RCPSP using CBR; Section 4 evaluates the proposed framework through some experiments; Section 5 surveys related work; finally, Section 6 draws some conclusions and suggests future work.

2 The Resource-Constrained Project Scheduling Problem

In project scheduling, a *project* consists of a set of *activities* that have to be carried out in order to finish the project. Every activity has certain *duration*. Some of these activities require that some other activities are already finished (precedence relationships), and some of them require the availability of certain *resources*. The outcome of project scheduling is a specification of the start(or finish) times of every activity —a schedule— so that the project can be completed and all the project constraints are met. Optimization functions are desirable to select the best among all possible schedules; for example, the *minimum time span* function is used to minimize projects completion.

Project scheduling is easy if only precedence relationships constrain the project: the Critical Path Method [4] provides allowable time windows for the activities of a project in polynomial time. However, scheduling becomes much harder when required resources are available in limited amounts, so that some tasks compete for the same resources. This class of problems is referred to as the Resource-Constrained Project Scheduling Problem (RCPSP), and is known to be NP-hard [3].

The single-mode Resource-Constrained Project Scheduling Problem for renewable resources can be defined as follows:

A **project** is a tuple $\langle \mathcal{J}, \mathcal{K} \rangle$ consisting of:

- A set $\mathcal{J} = \{0, 1, ..., J, J+1\}$ of **activities** to be executed, where J is the real number of activities, and activities 0 and $J+1$ are dummy activities used to represent the *project start* and *project end* respectively.
- A set $\mathcal{K} = 1, ..., K$ of **resources** available, where K is the number of constrained resources (non constrained resources are ignored).

For every **activity** $j \in \mathcal{J}$ there is a tuple $\langle d_j, \mathcal{P}_j, \mathcal{R}_j \rangle$ consisting of:

- A non-preemptable **duration** d_j, which specifies the amount of time required to complete the activity. Non preemptable means once started an activity cannot stop before completion.

- A set $\mathcal{P}_j \subseteq (\mathcal{J} - j)$ of **predecessors**. Activity i is predecessor of activity j ($i \in \mathcal{P}_j$) if the former has to be finished before the later can be started.
- A vector $\mathcal{R}_j = \langle r_{j,1}, ..., r_{j,K} \rangle$ of **resource requirements**. Each requirement $r_{j,k}$ represents the capacity of resource k required by activity j to be executed.

Every **resource** $k \in \mathcal{K}$ has a limited resource capacity of R_k. This capacity is reduced when being used by an activity, but it is restored when the activity ends.

The start and end activities are not real activities of the project: they do not take up time and do not require resources; however, they are included in the model for convenience, since they are useful for describing and implementing algorithms (actually, this has become a *de facto* standard representation).

The parameters d_j, $r_{j,k}$, and R_k are assumed to be deterministic; they remain the same for the entire duration of the project. These parameters are expressed discretely, using positive integers.

The objective of the RCPSP is to satisfy precedence- and resource-feasible completion times for all activities such that the timespan of the project is minimized. Since activities have deterministic durations, we may indistinctly use start times or finish times. Let f_j denote the finish time of activity j. A schedule $\mathcal{S}$ is given by a vector of finish times $\langle F_1, F_2, ..., F_J \rangle$. Let $\mathcal{A}(t) = j \in \mathcal{J} | F_j - d_j \leq t < F_j$ be the set of activities which are being processed (active) at time instant t. We can now specify the conceptual decision model (1) – (4) [5] for the RCPSP.

$$Min\ F_{n+1} \tag{1}$$

$$F_h \leq F_j - d_j \qquad j = 1, ..., J+1;\ h \in P_j \tag{2}$$

$$\sum_{j \in \mathcal{A}(t)} r_{j,k} \leq R_k \qquad k \in K;\ t \geq 0;\ r_{j,k} \leq R_k \tag{3}$$

$$F_j \geq 0 \qquad j = 1, ..., J+1 \tag{4}$$

The objective function (1) minimizes the finish time of the end activity and thus the make-span of the project; constraints (2) enforce the precedence relations between activities; and constraints (3) limit for each resource type k and each time instant t that the resource demand of the activities which are currently processed does not exceed the capacity. Finally, (4) defines the decision variables.

3 Case-Based Project Scheduling (CBPS)

3.1 Preliminary Notions: The Serial Schedule Generation Scheme

Most heuristic methods to solve the RCPSP are based on a *schedule generation scheme* (SGS), which starts from scratch and builds a feasible schedule by stepwise extension of a partial schedule. There are two different schedule generation schemes: Serial and Parallel. The Serial SGS (SSGS) performs activity increments (a new activity is scheduled each step), while the Parallel SGS (PSGS) performs time increments (time is incremented by one each step).

The SSGS consists of $g = 1, ..., J$ stages. At each stage one activity is selected and scheduled at the earliest precedence- and resource-feasible completion time. Associated with each stage g are two disjoint activity sets. The scheduled set $\mathcal{S}_g$ comprises the activities which have been already scheduled, the eligible set $\mathcal{D}_g$ comprises all activities which are eligible for scheduling. Let $\tilde{R}_k(t) = R_k - \sum_{j \in \mathcal{A}(t)} r_{j,k}$ be the remaining capacity of resource type k at time instant t and let $F_g = \{F_j | j \in S_g\}$ be the set of all finish times. Further let $\mathcal{D}_g = \{j \in \mathcal{J} \setminus \mathcal{S}_g \mid \mathcal{P}_j \subseteq \mathcal{S}_g\}$ the set of eligible activities. Given these definitions, we can now describe the SSGS algorithm as follows.]

Algorithm 1. The SSGS algorithm

initialization: $F_0 = 0, \mathcal{S}_0 = 0$
for $g = 1$ **to** J **do**
  Compute $D_g, F_g, \tilde{R}_k(t)(k \in K; t \in F_g)$
  Select one $j \in D_g$
  $EF_j = max_{h \in P_j}\{F_h\} + d_j$
  $F_j = min\{t \in [EF_j - d_j, LF_j - d_j] \cap F_g|$
  $r_{j,k} \leq \tilde{R}_k(\tau), k \in K, \tau \in [t, t + d_j[\cap F_g\} + d_j$
  $\mathcal{S}_g = \mathcal{S}_{g-1} \cup \{j\}$
  $F_{n+1} = max_{h \in \mathcal{P}_{n+1}}\{F_h\}$
end for
return F

The initialization assigns the start activity $j = 0$ a completion time of 0 and puts it into the partial schedule. At the beginning of each step g the decision set $\mathcal{D}_g$, the set of finish times $\mathcal{F}_g$ and the remaining capacities $\tilde{R}_k(t)$ at the finish times $t \in \mathcal{F}_g$ are computed. Afterwards, one activity j is selected from the decision set. The finish time of j is calculated by first determining the earliest precedence-feasible finish time EF_j and then calculating the earliest resource-feasible finish time F_j within $[EF_j, LF_j]$, where LF_j denotes the latest finish time as calculated by backward recursion [6] from an upper bound of the project finish time T.

The SSGS always generates feasible schedules which are optimal for the resource-unconstrained scheduling problem (1), (2), (4). Time complexity of SSGS is $O(n^2 \cdot K)$ [7]. The quality of the solutions obtained by the SSGS highly depends on the mechanism used to select the next activity $j \in D_g$ to be scheduled. Typically, this mechanism is based on the use of priority rules. A *priority rule* is a mapping which assigns each activity j in the decision set D_g a priority value $v(j)$ and an objective stating whether the activity with the minimum or the maximum value is preferred. In case of ties, one or several tie breaking rules have to be employed (e.g. choose the activity with the smallest activity label). For a description of different priority rules see for example [8].

A useful way of representing a schedule is an *activity list*: a list $\lambda = \langle j_1, j_2, ..., j_n]$ of activities that are precedence-feasible, i.e. we have $\mathcal{P}_{j_g} \subseteq \{j_1, ..., j_{g-1}\}$ where $g \in 1, 2, ..., n$. An activity list containing all the activities of a project encodes a schedule for that project. There is a version of the SSGS for activity lists, which

simply selects the next activity to be scheduled each step as the next activity in the activity list. Formally, if we use j_g to denote the g^{th} activity of λ, then in the SSGS (Algorithm 1) we simply have to replace 'Select one $j \in D_g$' (line 4) with 'Select $j = j_g$', and remove D_g from the initialization step [9].

3.2 Overview of the Framework

Our proposal to solve the RCPSP uses the SSGS as the underlying mechanism to ensure that the solutions obtained meet the precedence and resource capacity constraints. In particular, we introduce the notion of *generalized activity list* (GAL), which results of removing the precedence condition from the definition of an activity list. A GAL also encodes a solution schedule, one which can be obtained by a modified version of the SSGS: the *SSGS for generalized activity lists*, abbreviated SSGS-GAL, and described later as Algorithm 4.

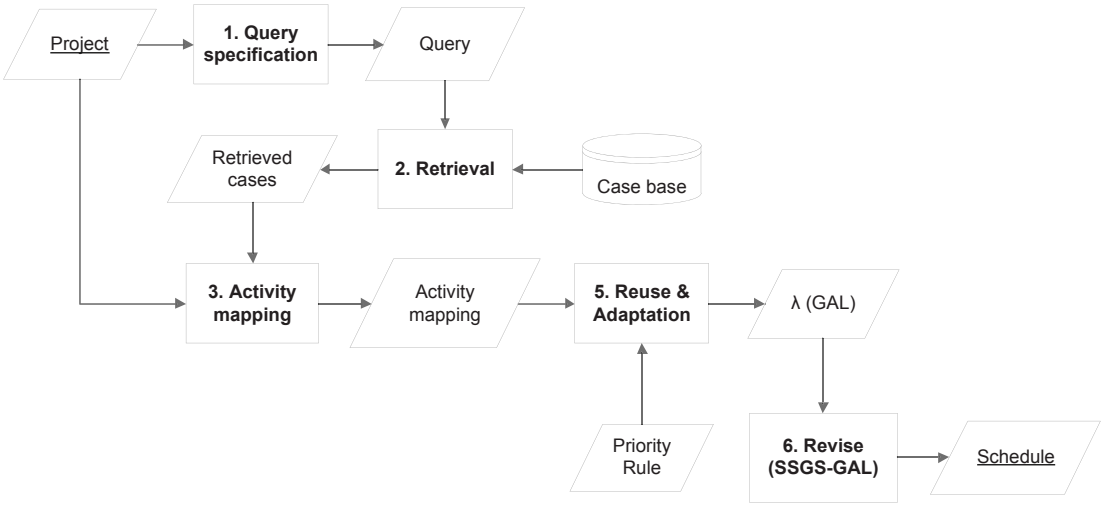

Fig. 1. Process of building a case-based activity list

Figure 1 shows a flowchart of the overall process. The process starts with a project to be scheduled, a priority rule and a case-base, and produces a complete and valid schedule for the project. The *case-base* contain resource-constrained projects and activity lists that solve those projects. This framework contains the usual stages in a CBR system with one remarkable difference : the retrieval step is divided in two phases. During the retrieval step some cases are retrieved using a standard similarity measure computed over global project properties, as usual, but there is an additional retrieval step (the activity mapping) that uses a graph-based similarity measure to compare projects at the structural level. Next sections describe the entire method step by step.

3.3 Query Specification and Case Retrieval

Query specification and case retrieval are tightly related, the former analyses the target project to find the values of some properties that are then used by

the later to retrieve cases containing projects that are similar the target project. These properties are specified as attribute-value pairs, a representation that supports a wide range of well known and very fast similarity measures and retrieval algorithms. In particular we use the classical k-Nearest Neighbour (k-NN) algorithm and compute project similarity as a weighted average of similarity measures for certain global project properties summarized below.

Network complexity (NC): the average number of (non redundant) arcs per node (including super-source and sink)

Resource Factor (RF): reflects the average portion of resources requested per activity. $RF = 1$ means each activity requests all resources, while $RF = 0$ indicates the opposite, which corresponds to the unconstrained case. For the single mode RF is calculated as follows:

$RF = \frac{1}{J}\frac{1}{|\mathcal{K}|}\sum_{j=1}^{J}\sum_{k\in\mathcal{K}} Q_{j,k}; Q_{j,k} = \{1 \text{ if } r_{jk} > 0 \text{ ; } 0 \text{ otherwise}\}$

Resource Strength (RS): aims at expressing how difficult it is to satisfy the resource constrains of a project. Several ways to compute RS have been proposed. We have chosen the one used to generate the PSPLIB datasets, because it seems to capture better this notion of difficulty, and PSPLIB is the preferred benchmark to compare scheduling algorithms. RS is a scaling parameter in the interval $[0, 1]$ used to compute resource availability (R_k) as a convex combination of a minimum and a maximum level, R_k^{min} and R_k^{max} respectively. For the single mode, R_k is defined as follows:

$$R_k = R_k^{min} + round(RS \times (R_k^{max} - R_k^{min}))$$

There is no exact procedure to isolate RS in the former equation, but it can be approximated by *numerical methods*: computing lower and upper bounds for RS and interpolating between them.

3.4 Activity Mapping

The previous retrieval step obtains a list of cases containing similar projects, where project similarity is based on global project properties, which is appropriate to find a set of projects that are potential candidates for reuse. But in order to select the best candidate for reuse, we have to compare projects at a deeper level, considering the structural components of a project, that is, the network of activities, precedence relations and resource constraints.

In order to reuse a solution, we also need a mapping of activities between the project to be scheduled and the projects retrieved from the case base. Actually, computing the similarity and obtaining the activity mappings are part of the same process, since computing the structural similarity involves maximizing the set of activity mappings.

Mathematically, a project can be described by a DAG, where nodes correspond to activities, and edges correspond to precedence relations; therefore, computing the structural similarity of two projects becomes a problem of computing the similarity of two graphs. Typically, graph similarity algorithms are variations or derivatives of the *maximum common subgraph isomorphism* (MCS) problem, which takes two graphs G1 and G2 as input, and aims at finding the largest induced subgraph of G1 isomorphic to a subgraph of G2. Exact MCS algorithms compute several alternative mappings of nodes in G1 that are isomorphic to

nodes in G2, and return the maximal set of node mappings. If we use nodes to represent project activities, then the maximal node mapping returned by an MCS algorithm is precisely the activity mapping we are seeking.

The MCS isomorphism problem is NP-hard, so exact algorithms do not scale well with the size of the problem. There are also approximate algorithms reported in the literature and available as free software libraries, like Absurdist II [10]. Nevertheless, our experience with this algorithm has not been satisfactory, since the quality of the solutions obtained was rather poor.

Everything considered, we have developed our own algorithm to compute the similarity of two projects. This is an approximate algorithm in two ways: on the one hand, it does not enforce the isomorphism condition to map nodes; on the other hand, it does not compute all possible node mappings. In particular, our algorithm reduces the number of potential activity mappings by using heuristics based on some of the features making a project a special case of a DAG, namely: (a) any project has a single start node and a single end node (the dummy start/end activities), and (b) activities are pre-sorted taking into account precedence relations: activity j can not be a predecessor of an activity k given $k < j$.

Let us introduce some notation and definitions needed to describe the activity mapping algorithm.

$Q = \langle \mathcal{J}^Q, \mathcal{K}^Q \rangle$ is the target project, the one to be scheduled.
$P = \langle \mathcal{J}^P, \mathcal{K}^P \rangle$ is a project retrieved from the case base.
$S \in [0, 1]$ is the similarity between projects P and Q
M and M' are sets of activity mappings between $\mathcal{J}^P$ and $\mathcal{J}^Q$. An activity mapping is a pair of activities (j, j'), where $j \in \mathcal{J}^P$ and $j' \in \mathcal{J}^Q$.
$sim(j, j')$ is a function that returns the similarity of $j \in \mathcal{J}^P$ and $j' \in \mathcal{J}^Q$.

Algorithm 2 is a pseudocode description of the algorithm for computing the similarity between two projects and obtaining a mapping of activities.

First (lines 2 to 6), the similarity of every possible pair of activities is computed, and the pairs with a similarity greater than certain threshold α are added to the set of potential activity mappings M'. Activity similarity measures are described later.

Next (lines 7 to 13), for each activity j' in Q, a single activity mapping (h, j') from M' is selected and included in the final set M of mappings. The selection of the final activity mappings uses a simple heuristic: the mappings comprising the most similar activities are preferred, and in case of ties, the mapping whose left activity h has a lower identifier is selected. This is an approximate procedure, since not all possible combinations of activity mappings are considered, but only one.

Finally (lines 14 to 17), project similarity S is computed as the sum of similarities for all activities in the final set of activity mappings, divided by the number of activities in the target project ($\mathcal{J}^Q$). Since the similarity between activities is a value in [0,1], and the maximum number of activity mappings is precisely $\mathcal{J}^Q$, then project similarity is also a value in [0,1].

We still have to define a similarity measure for activities. The results of the algorithm and its complexity can be strongly influenced by the way we compute similarity between activities. There are two aspects to take into account to assess the similarity between two tasks: resource requirements and precedence relations.

Algorithm 2. Activity Mapping Algorithm

initialization: $M = \emptyset$; $M' = \emptyset$; $S = 0$
for all $j \in \mathcal{J}^P$, $j' \in \mathcal{J}^Q$ **do**
  if $sim(j, j') > \alpha$ **then**
    $M' = M' \cup (j, j')$
  end if
end for
for all $(j, j') \in M'$ **do**
  if $\nexists h : (h, j') \in M'$ **then**
    $M = M \cup (j, j')$
  else if $sim(j, j') > sim(h, j') \vee (sim(j, j') = sim(h, j') \wedge j < h)$ **then**
    $M = (M \setminus (h, j')) \cup (j, j')$
  end if
end for
for all $(j, j') \in M$ **do**
  $S = S + sim(j, j')$
end for
$S = S \ / \ |\mathcal{J}^Q|$
return M, S

We have adopted a binary similarity approach to compute activity similarity, that is, activity similarity can take only two values: 0 or 1. Conceptually, we define activity mapping using equivalence relations, as follows:

Two activities are *resource-equivalent* if there is a mapping of resources such that all pairs of resource requirements are *capacity-proportional*.

Two resource requirements are *capacity-proportional* if they amount for the same proportion of resource capacity.

Two activities are *equivalent* if they are resource-equivalent and there is a mapping of successors such that all pairs of successors are resource-equivalent.

Now we can define the similarity of two tasks j and j':

(a) $sim(j, j') = 1 \iff j$ and j' are *equivalent*; otherwise $sim(j, j') = 0$

The similarities obtained by the mapping algorithm are used to select the case with the highest similarity S, which is referred as the **best case** henceforth. The mapping M of activities between the target project and the project included in the best case are passed to the next step to produce an activity list, as described in the following section.

3.5 Reuse and Adaptation

The previous step obtains a mapping of activities between the target project and the project in the best case. The ordering of mapped activities in the solution of the best case provides a tentative ordering of activities in the new project to build the required schedule. In order to produce a complete activity list we have conceived a method that uses both the schedule from the best case and a standard priority rule, as described in Algorithm 3.

$\mathcal{J}^Q$ and $\mathcal{J}^P$ represent the sets of activities in the target project Q and the project P stored in the best case respectively. Let M be the set of activity

Algorithm 3. Reuse and Adaptation Algorithm

1: **initialization**: $\theta = \emptyset$
2: **for all** $j \in \mathcal{J}^Q$ **do**
3: **if** $\exists(j', j) \in M : j' \in \mathcal{J}^P$ **then**
4: $\theta = \theta \cup j$
5: **end if**
6: **end for**
7: $\lambda = SORT(\mathcal{J}^Q, \theta, \psi)$
8: **return** λ

mappings between $\mathcal{J}^Q$ and $\mathcal{J}^P$ and let ψ denote a standard priority rule. First the algorithm computes a partial activity list θ (lines 2 to 6) by traversing $\mathcal{J}^Q$ and adding to θ those activities that are mapped to activities in $\mathcal{J}^P$. Finally, the algorithm calls the $SORT$ function (line 7) to produce λ, a complete activity list that encodes a solution for Q. $SORT$ takes $\mathcal{J}^Q$, θ and ψ as inputs, and obtains λ as output by applying the following order relation on $\mathcal{J}^Q$: if two activities $i, j \in \mathcal{J}^Q$ are both included in θ then their order is kept the same in λ, otherwise their order is obtained by applying priority rule ψ.

However, being the mapping of activities approximate, the ordering of activities obtained from the best case's solution does not guarantees the satisfaction of Q's constraints, so θ is actually a partial GAL. As a consequence, the resulting list λ is also a GAL.

3.6 Revise

The reuse step returns a partial GAL (λ), so the precedence and resource-capacity conditions are not guaranteed. As a consequence, we can not use the standard SSGS for activity lists to obtain the solution encoded by λ, since it might produce an incorrect schedule. In order to obtain a valid schedule we have developed a new version of the SSGS for generalized activity lists, abbreviated SSGS-GAL, which is described in Algorithm 4, where $g(j) \in \{1, 2, ..., J\}$ denotes the position of j in λ. A solution obtained by the SSGS-GAL is guaranteed to be correct even though the activity list used as input is not, since the project constraints are enforced by the SSGS-GAL when computing the decision set D_g (Line 3).

Algorithm 4. The SSGS algorithm for generalized activity lists (SSGS-GAL)

1: **initialization**: $F_0 = 0, \mathcal{S}_0 = 0$
2: **for** $g = 1$ **to** n **do**
3: Compute $D_g, F_g, \tilde{R}_k(t)(k \in K; t \in F_g)$
4: Select $j \in D_g | \forall j' \in D_g : g(j) > g(j')$
5: $EF_j = max_{h \in P_j}\{F_h\} + d_j$
6: $F_j = min\{t \in [EF_j - d_j, LF_j - d_j] \cap F_g | r_{j,k} \leq \tilde{R}_k(\tau), k \in K, \tau \in [t, t + d_j[\cap F_g\} + d_j$
7: $\mathcal{S}_g = \mathcal{S}_{g-1} \cup \{j\}$
8: $F_{n+1} = max_{h \in \mathcal{P}_{n+1}}\{F_h\}$
9: **end for**
10: **return** F

The SSGS-GAL, as the SSGS, needs a priority rule to compute the decision set (D_g).

4 Experimental Evaluation

In order to evaluate a learning approach such as CBR, one should demonstrate its ability to generalize, that is, the learning method should be able to solve new problems that were not included in the learning dataset. In order to have a pool of datasets for learning and testing purposes we have taken the well known Patterson's dataset [11] , and we have used it as a source to create new datasets by randomly modifying its problems.

In particular, we have created two groups of new datasets: On the one hand, **PatI5** comprises 3 data sets obtained by removing between 1 and 5 random activities from Patterson's original problems. On the other hand, **PatD** comprises a sequence of 9 interdependent data sets; the first data set in the sequence (PatD1) results of removing 1 random job from projects in the Patterson's data set, the next data set (PatD2) results of removing 1 random activity from the previous data set in the sequence (thus accumulates 2 removals from the original data set), and so on.

In order to compare different methods, we have first obtained the optimal solutions for the new data sets using the *branch and bound* algorithm proposed by [12]; that way, we can compare algorithms by comparing their solutions with the optimal solution.

We have performed two groups of experiments: In the first group, we assess the impact of k and the size of the case base on the performance of the Case-Based Project Scheduling (CBPS) algorithm. In the second group we assess how the differences between the learning and the test data sets impact the performance of the CBPS algorithm.

In our experiments, we use the *average relative error* to measure the performance of a scheduling algorithm. The relative error of a single project is computed as $(timespan - optimalTimespan)/optimalTimespan$, thus the lower the error the better. In all the experiments that follow we include a comparison with a single-pass SSGS using LST (Latest Start Time) as the priority rule. Correspondingly we have also used LST as the priority rule ψ to complete the partial activity list obtained from the best case, as described in Algorithm 3.

Impact of k and the Size of the Case Base. The aim here is to study how the size of the case base and the number of cases retrieved (k) influence the performance of the CBPS algorithm. We expect CBPS to obtain better solutions as the size of the case base and the number of cases retrieved increase.

Table 1 shows the results for the PatI5 data sets using different case bases, where CBPS:k denotes the CBPS algorithm with k being the number of cases retrieved. data sets with suffix 'a' and 'b' used for learning, while the one with suffix 'c' is used for testing purposes. CBPS obtains better results than LST in all the scenarios, and the results improve consistently with the number of cases retrieved . First and second columns show results when using PatI5a and Pat5b respectively to build the case base. Since they have the same size, results

Table 1. Average relative errors for PaIt5 data sets using different case bases

Algor. \ CB	PatI5a	PatI5b	PatI5a ∪ PatI5b	PatI5c
LST	0,090	0,090	0,090	0,090
CBPS:1	0,083	0,085	0,078	0,006
CBPS:3	0,074	0,084	0,073	0,000
CBPS:5	0,073	0,082	0,069	0,000
CBPS:10	0,068	0,073	0,064	0,000
CBPS:20	0,066	0,070	0,058	0,000

are quite similar. Third column shows results when using PatI5a and PatI5b together for the case base; as expected, we obtain better results due to the increase in the size of the case base. The last column is provided to check out that the algorithm works properly: since we use the same data set for learning and testing, the CBPS algorithm should be able to retrieve exactly the project being solved, and thus it should provide the optimal solution, i.e. error would be zero. As observed, CBPS is able to find the optimal solutions when $k \geq 3$, and the error when retrieving a single case ($k = 1$) as as low as 0.06. These result suggest that the global project characterization (network complexity, resource factor and resource strength) is a good discriminant between projects.

Impact of the Degree of Difference between the Learning Data Sets and the Test Data Sets. The experiments reported here study the impact of the degree or amount of difference between the learning data set and test data set. We expect CBPS to obtain worse results as the amount of difference increases.

Table 2. Average relative errors for PatD data sets, with Pat5 as the case-base

Algorithm	Pat1	Pat2	Pat3	Pat4	**Pat5**	Pat6	Pat7	Pat8	Pat9
LST	0,096	0,096	0,099	0,091	0,093	0,085	0,080	0,079	0,071
CBPS:1	0,097	0,099	0,085	0,066	0,006	0,047	0,068	0,067	0,059
CBPS:3	0,095	0,096	0,076	0,052	0,000	0,033	0,054	0,059	0,071
CBPS:5	0,091	0,095	0,073	0,047	0,000	0,030	0,052	0,060	0,071
CBPS:10	0,095	0,091	0,075	0,046	0,000	0,025	0,047	0,050	0,062
CBPS:20	0,092	0,085	0,069	0,046	0,000	0,027	0,037	0,043	0,058

Table 2 shows results when solving PatD data sets using Pat5 as the case base. As expected, Pat5 is solved optimally for a small k (when $k \geq 3$). In Pat1,...,Pat4 the projects to be solved have more activities than the projects in the case-base (Pat1 has 4 more activities, Pat2 has 3 more activities, etc.), so only a portion of the activities could potentially be mapped to activities in a retrieved project. The opposite happens to projects in Pat6 to Pat9, which have fewer activities than projects in the case base. Therefore we expect CBPS to behave much better when solving the later than the former; which is actually confirmed by the experiments. In all the tests CBPS obtained better solutions than LST. Notice that the fewer the differences between the test and learning data sets, the comparatively better our algorithm behaves.

5 Related Work

The first system experimentally evaluated is CABINS [13], a framework for iterative schedule repair based on user optimization preferences. CABINS uses a a case-based approach for capturing human experts' preferential criteria about scheduling quality and control knowledge to speed up schedule revision. Through iterative schedule repair, CABINS improves the quality of sub-optimal schedules by using past repair experiences for (1) repair action selection, (2) evaluation of intermediate repair results, and (3) recovery from revision failures. Extensive experimentation on a job-shop scheduling domain shows an improvement in the efficiency of the revision process while preserving the quality of the resultant schedule. The first constructive CBR approaches for scheduling including experimental evaluation were reported by [14,15]. The first paper describes a CBR method to solve the Traveling Salesman Problem (TSP) by reusing and adapting complete solutions. The proposed method, called T-CBR, was compared against Simulated Annealing (SA) and a Myopic algorithm, concluding that SA produces the best solutions but takes the longest, the Myopic algorithm is the faster algorithm but produces the worst solutions, and T-CBR produces medium quality solutions considerably faster than SA. The second paper compares two CBR methods for solving the *job-shop* scheduling problem with a single machine and sequence dependent setup times. CBR has also been proposed to deal with *workforce rostering* problems [16,17]. Cases represent rostering constraints and generalized patterns of workforce allocation (shift patterns) that satisfy those constraints. In this proposal CBR is used in a constructive way to build up a schedule that is then fixed using rules to remove constraint violations. Fixes are expressed as ordered series of shift swapping rules. This separation between the roster generation using CBR and the fix mechanisms is well suited to perform reactive scheduling with no extra effort: the very same mechanisms used to fix a new roster can be used to fix rosters that have subsequently become broken. Typically, scheduling problems the proposals introduced above adopt the classical representation of cases as lists of attribute-value pairs. A radically different approach using graphs was proposed in [18] to deal with timetabling problems. Experimental results showed the effectiveness of the retrieval and adaptation steps. This method was further developed and evaluated across a wider range of problems [19]. Later, a multiple-retrieval approach was proposed that partitions a large problem into small solvable sub-problems [20].

CABAROST (CAse-BAsed Rostering) is a repair-based method for solving nurse rostering problems [21,22]. CABAROST retrieves previously encountered constraint violations and reuses their repairs to solve new rostering problems.

All in all, research on the application of Case-Based Reasoning (CBR) to scheduling is scarce and limited in scope. On the one hand, the application domain has been limited to very specific forms of scheduling, such as the job-shop problem, rostering problems, and time-tabling problems. On the other hand, really few systems have been evaluated experimentally. Finally, in several proposals CBR is not the actual scheduling method, but just a complementary tool used either to select the actual scheduling method or to configure it (for example to choose one amongst a number of alternative heuristic rules).

6 Conclusions and Future Work

Compared with other attempts to apply CBR to scheduling problems, we have addressed a more general class of problems. In fact, a wide area of combinatorial problems are known to be special cases of the RCPSP, including: the *two-dimensional cutting stock* problem, the *bin packing* problem, and production scheduling problems such as the *job-shop* scheduling problem, the *flow-shop* problem and the *open-shop* problem.

In this paper, we describe a Case-Based Reasoning (CBR) framework to solve the Resource-Constrained Project Scheduling Problem (RCPSP) in a constructive way, using past schedules as a starting point to build new schedules. The aim is not to compete with the best results reported in the literature by special purpose specific approaches, but to generate competitive results across a wider range of problems: for example, problems with poorly defined domains or high degrees of uncertainty. Note that in general real scheduling problems are much more complex and uncertain than the mathematical models used in academic research. The more complex and uncertain the domain, the more appropriate CBR would be compared with theoretical approaches. Besides, a CBR approach such as CBPS will be able to adapt and improve results over time by storing new cases in the case base and removing old ones. The experience obtained from real world examples will probably capture complexities of the domain that will be much harder to represent in a theoretical model.

In the experiments performed so far we have consistently obtained better results than using the best simple-pass priority-rule based heuristics. These results demonstrate the feasibility of our proposal and indicate that under reasonable assumptions this approach may become a useful tool to trade-off quality of the solutions and efficiency, which is a requirement to address real world problems.

Our framework extends the well-known Serial Schedule Generation Scheme by introducing the SSGS for generalized activity lists (SSGS-GAL). By using SSGS as the underlying validation mechanism we ensure that only valid solutions are generated, together with other interesting properties. Our method is compatible with the standard view of heuristic priority rules as functions that compute a priority value for all activities of a project. This approach supports the use of single-pass SSGS methods as well as most multi-pass SSGS methods, including *multi-priority rule* methods, *forward-backward* scheduling methods, and *sampling* methods. Furthermore, our framework is flexible enough to support a wide range of similarity measures to compare projects, which opens many possibilities to tune up the algorithm and adapt it to different circumstances.

A limitation of our approach is the fact that in order to work properly, CBR requires good cases, that is, cases that are representative of the application domain, and cases that have good solutions. In the real world, sometimes there may be a repository of past projects which can be used to generate an initial case base, but that will not be true in general. When no previous experience is available, one has to generate new cases using another (non-CBR) method, and therefore, the quality of the solutions obtained by CBR would depend upon the quality of the solutions obtained by the non-CBR method. In other words, if there is no previous experience it would probably be better to use a non-CBR approach, at least until gaining experience.

There is a number of ways to further expand our framework. Firstly, there is plenty of room to experiment with different similarity metrics to retrieve projects. Secondly, it would be possible to experiment with different scheduling algorithms, like the Parallel Scheduling Generation Scheme and some multi-pass methods (eg. sampling and multi-rule methods). Another interesting extension would be the building of new schedules by combining portions of multiple solutions, instead of reusing a single solution from the case-base.

Acknowledgments. Support for this work came from project Next-CBR MICIN TIN2009-13692-C03-01.

References

1. Herroelen, W., De Reyck, B., Demeulemeester, E.: Resource-constrained project scheduling: a survey of recent developments. Computers and Operations Research 25(4), 279–302 (1998)
2. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(1), 3–41 (1999)
3. Blazewicz, J., Lenstra, J., Kan, A.R.: Scheduling subject to resource constraints classification and complexity. Discrete Applied Mathematics 5, 11–24 (1983)
4. Kelley, J.: The Critical-Path Method: Resources Planning and Scheduling. In: Industrial Scheduling, pp. 347–365 (1963)
5. Christofides, N., Alvarez-Valdés, R., Tamarit, J.: Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research 29(3), 262–273 (1987)
6. Elmaghraby, S.: Activity networks: Project planning and control by network models. John Wiley & Sons, New York (1977)
7. Pinson, E., Prins, C., Rullier, F.: Using tabu search for solving the resource-constrained project scheduling problem. In: Proceedings of the 4th International Workshop on Project Management and Scheduling, pp. 102–106 (1994)
8. Alvarez-Valdés, R., Tamarit, J.: Heuristic algorithms for resource-constrained project scheduling. In: Advances in Project Scheduling, pp. 113–134. Elsevier (1989)
9. Kolisch, R., Hartmann, S.: Heuristic Algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Handbook of Recent Advances in Project Scheduling, pp. 147–178. Kluwer Academic Publishers (1999)
10. Feng, Y., Goldstone, R.L., Menkov, V.: A graph matching algorithm and its application to conceptual system translation. International Journal on Artificial Intelligence Tools 14, 77–79 (2004)
11. Patterson, J.H.: A comparison of exact procedures for solving the multiple constrained resource project scheduling problem. Management Science, 854–867 (1984)
12. Sprecher, A., Drexl, A.: Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. European Journal of Operational Research 107(2), 431–450 (1998)
13. Miyashita, K., Sycara, K.P.: Cabins: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. Artificial Intelligence 76(1-2), 377–426 (1995)

14. Cunningham, P., Smyth, B., Hurley, N.: On the Use of CBR in Optimisation Problems Such as the TSP. In: Aamodt, A., Veloso, M.M. (eds.) ICCBR 1995. LNCS, vol. 1010, pp. 401–410. Springer, Heidelberg (1995)
15. Cunningham, P., Smyth, B.: Case-based reasoning in scheduling: Reusing solution components. International Journal of Production Research 35(11), 2947–2962 (1997)
16. Scott, S., Simpson, R., Ward, R.: Combining case-based reasoning and constraint logic programming techniques for packaged nurse rostering systems. In: Proceedings of the Third UK Case-Based Reasoning Workshop (1997)
17. Scott, S., Simpson, R.M.: Case-Bases Incorporating Scheduling Constraint Dimensions - Experiences in Nurse Rostering. In: Smyth, B., Cunningham, P. (eds.) EWCBR 1998. LNCS (LNAI), vol. 1488, pp. 392–401. Springer, Heidelberg (1998)
18. Burke, E.K., MacCarthy, B., Petrovica, S., Qu, R.: Structured cases in case-based reasoning: reusing and adapting cases for time-tabling problems. Knowledge-Based Systems 13(2-3), 159–165 (2000)
19. Burke, E.K., MacCarthy, B.L., Petrovic, S., Qu, R.: Case-Based Reasoning in Course Timetabling: An Attribute Graph Approach. In: Aha, D.W., Watson, I. (eds.) ICCBR 2001. LNCS (LNAI), vol. 2080, pp. 90–104. Springer, Heidelberg (2001)
20. Burke, E., MacCarthy, B., Petrovic, S., Qu, R.: Multiple-retrieval case-based reasoning for course timetabling problems. Journal of the Operational Research Society 57(2), 148–162 (2006)
21. Beddoe, G., Petrovic, S.: A novel approach to finding feasible solutions to personnel rostering problems. In: Proceedings of the 14th Annual Conference of the Production and Operations Management Society (April 2003)
22. Petrovic, S., Beddoe, G., Vanden Berghe, G.: Storing and Adapting Repair Experiences in Employee Rostering. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 148–165. Springer, Heidelberg (2003)
23. Beddoe, G., Petrovic, S.: Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. European Journal of Operational Research 175(2), 649–671 (2006)
24. Beddoe, G., Petrovic, S.: Enhancing case-based reasoning for personnel rostering with selected tabu search concepts. Journal of the Operational Research Society 58(12), 1586–1598 (2007)
25. Beddoe, G., Petrovic, S., Li, J.: A hybrid metaheuristic case-based reasoning system for nurse rostering. Journal of Scheduling, 99–119 (2008)
26. Burke, E.K., Petrovic, S., Qu, R.: Case-based heuristic selection for timetabling problems. Journal of Scheduling 9(2), 115–132 (2006)
27. Petrovic, S., Yang, Y., Dror, M.: Case-based selection of initialisation heuristics for metaheuristic examination timetabling. Expert Systems with Applications 33(3), 772–785 (2007)
28. Kolisch, R.: Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 90(2), 320–333 (1996)
29. Sprecher, A., Kolisch, R., Drexl, A.: Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research 80(1), 94–102 (1995)