User recommendations for the optimized execution of business processes

Irene Barba [a,*], Barbara Weber [b], Carmelo Del Valle [a], Andrés Jiménez-Ramírez [a]

[a] Departamento de Lenguajes y Sistemas Informáticos, University of Seville, Spain
[b] Department of Computer Science, University of Innsbruck, Austria

Keywords:
Business processes
Workflow management
Business intelligence
Flexible process-aware information systems
Recommendations
Constraint programming
Scheduling

A B S T R A C T

In order to be able to flexibly adjust a company's business processes (BPs) there is an increasing interest in flexible process-aware information systems (PAISs). This increasing flexibility, however, typically implies decreased user guidance by the PAIS and thus poses significant challenges to its users. As a major contribution of this work, we propose a recommendation system which assists users during process execution to optimize performance goals of the processes. The recommendation system is based on a constraint-based approach for planning and scheduling the BP activities and considers both the control-flow and the resource perspective. To evaluate the proposed constraint-based approach different algorithms are applied to a range of test models of varyingcomplexity. The results indicate that, although the optimization of process execution is a highly constrained problem, the proposed approach produces a satisfactory number of suitable solutions.

1. Introduction

Nowadays, there exists a growing interest in aligning information systems in a process-oriented way [9,55]. In the current dynamic business world the economic success of an enterprize increasingly depends on its ability to react to changes in its enterprize in a quick and flexible way [23]. Therefore, flexible process-aware information systems (PAISs) [54,33,32] are required to allow companies to rapidly adjust their business processes (i.e., set of related structured activities whose execution produces a specific service or product required by a particular customer) to changes in the environment [45]. The specification of process properties in a declarative way is an important step towards the flexible management of PAISs [47]. The advantages of using declarative languages for business process (BP) modeling instead of imperative languages, i.e., support for partial workflows [52], absence of over-specification [30], and provision of more maneuvering room for end users [30], are discussed in several studies (e.g., [39,11,12]).

Due to their flexible nature, frequently several ways to execute declarative process models exist. Typically, given a certain partial trace (reflecting the current state of the process instances), users can choose from several enabled activities which activity to execute next. This selection, however, can be quite challenging since performance goals of the process (e.g., minimization of overall completion time) should be considered, and users often do not have an understanding of the overall process. Moreover, optimization of performance goals requires that resource capacities are considered. Therefore, recommendation support is needed during BP execution, especially for inexperienced users [49].

In order to support users of flexible PAISs during process execution in optimizing performance goals like minimizing the overall completion time (i.e., time needed to complete all process instances which were planned for a certain period), we propose the generation of optimized enactment plans. For this, activities to be executed have to be selected and ordered (planning problem [18]) considering both the control-flow and resource constraints (scheduling problem [6])[1] imposed by the declarative specification. In

* Corresponding author. Tel.: +34 954559814.
E-mail addresses: irenebr@us.es (I. Barba), barbara.weber@uibk.ac.at (B. Weber), carmelo@us.es (C. Del Valle), ajramirez@us.es (A. Jiménez-Ramírez).

[1] In the current work the role-based allocation pattern [38], i.e., resources of a specific role are required for activity execution, is considered.

general, for the execution of BP activities, the use of shared resources is necessary, which must be managed in an effective way in order to optimize certain objectives of the BP enactment.

For planning and scheduling (P&S) the activities in a way that the process goal is optimized, a constraint-based approach is proposed since constraint programming (CP) [35] supplies a suitable framework for modeling and solving problems involving P&S aspects [40]. For this, the declarative model is complemented with information related to estimates regarding the number of instances, activity durations, and resource availabilities. Recommendations on possible next steps are then generated taking the partial trace and the optimized plans into account. Replanning is supported if actual traces deviate from the optimized enactment plans (e.g., because estimates turned out to be inaccurate).

In order to evaluate the effectiveness of the proposed constraint-based approach, different algorithms are applied to a range of test models of varying complexity. The suitability of our approach is tested regarding both (1) build-time, i.e., for the generation of complete optimized plans before starting the BP enactment; and (2) run-time, i.e., for the generation of partial, optimized plans by considering the actual partial trace of the process as the execution of the process proceeds (replanning). For both build-time and run-time evaluations, the results which are obtained by each algorithm are compared to obtain information about which algorithm is better under which circumstances and whether one algorithm is better in general. The results indicate that the proposed approach produces a satisfactory number of suitable solutions, i.e., solutions which are optimal in most cases and quite good in other cases.

Initial aspects related to the proposed constraint-based approach and how to use this approach as basis for creating user recommendations have been previously presented in [2] and [5] respectively. However, this paper significantly extends and improves this previous work by: (1) providing background on related areas (cf. Section 2), (2) giving more details about the method for generating recommendations, e.g., by including an algorithm for providing recommendations (cf. Section 3), (3) introducing new and efficient search algorithms for solving the constraint-based problems (cf. Section 4.3), (4) evaluating the effectiveness of the proposed approach over a set of representative problems for testing its suitability regarding both build and run-time, and therefore, demonstrating that it can work in practice for managing realistic problems (cf. Section 6), (5) analyzing the advantages and shortcomings of our approach (cf. Section 7), and (6) extending the related work which is analyzed (cf. Section 8).

This paper is organized as follows: Section 2 introduces the related areas, Section 3 includes an overview of our proposal, Section 4 details the method for generating the optimized enactment plans, Section 5 shows the application of the proposed approach to a running example, Section 6 shows some experimental results, Section 7 presents a critical discussion of our proposal, Section 8 summarizes related work, and finally, Section 9 includes conclusions and future work.

2. Background

Our work combines aspects of scheduling, planning, and CP to support users during the execution of flexible declarative BPs. Section 2.1 provides backgrounds regarding declarative processes. Section 2.2 gives an overview of planning, scheduling and CP.

2.1. Declarative BP models

Recently, constraint-based approaches have received increased interest [51,29], since they suggest a fundamentally different way of describing BPs which seems to be promising in respect to the support of highly dynamic processes [51,29]. Irrespective of the chosen approach, requirements imposed by the BPs need to be reflected by the process model. This means that desired behavior must be supported by the process model, while forbidden behavior must be prohibited [30,47,26]. While imperative process models specify exactly how things have to be done, declarative process models focus on what should be done. In the current approach, we consider declarative BP specifications since, as stated, the specification of process properties in a declarative way is an important step towards the flexible management of PAISs [47]. In the literature, several rule-based and constraint-based languages for declarative BP modeling are proposed (e.g., [46,8,53,25]). In our proposal we use ConDec [46] for the BP control-flow specification. We consider ConDec to be a suitable language, since it allows the specification of BP activities together with the constraints which must be satisfied for correct BP enactment and for the objective to be achieved. Moreover, ConDec allows the specification of a wide set of BP models of varied nature, flexibility and complexity in a simple way. ConDec is based on constraint-based BP models (cf. Definition 1), i.e., including information about (1) activities that can be performed as well as (2) constraints prohibiting undesired process behavior.

Definition 1. A *constraint-based process model* $S = (A,C)$ consists of a set of activities A, and a set of constraints C prohibiting undesired execution behavior.

The activities of a constraint-based process model can be executed arbitrarily often if not restricted by any constraints (for a description of the complete set of constraints, cf. [46]). ConDec constraints can be divided into 3 groups:

1. *Existence* constraints: unary relationships concerning the number of times one activity is executed. As an example, ExactlyN(A) specifies that A must be executed exactly N times.
2. *Relation* constraints: positive binary relationships used to establish what should be executed. As an example, Precedence(A,B) specifies that to execute activity B, activity A needs to be executed before.
3. *Negation* constraints: forbid the execution of activities in specific situations. As an example, NotCoexistence(A,B) specifies that if B is executed, then A cannot be executed, and vice versa.

Fig. 1 shows a simple constraint-based model which is composed by activities *A*, *B*, and *C*, and constraints *C*1 (ExactlyN(A)), *C*2 (Precedence(A,B)), *C*3 (Precedence(A,C)), and *C*4 (NotCoexistence(B,C)).

As the execution of a constraint-based model proceeds, information regarding the executed activities is recorded in an execution trace (cf. Definition 2).

Definition 2. Let $S=(A,C)$ be a constraint-based process model with activity set A and constraint set C. Then: A *trace* σ is composed by a sequence of starting and completing events $\langle e_1, e_2, \ldots e_n\rangle$ regarding activity executions a_i, $a \in A$, i.e., events can be:

1. $start(a_i, Rj_k, T)$, i.e., the i-th execution of activity a using k-th resource with role j is started at time T.
2. $comp(a_i,T)$, i.e., the i-th execution of activity a is completed at time T.

A process instance (cf. Definition 3) represents a concrete execution of a constraint-based model and its execution state is reflected by the execution trace.

Definition 3. Let $S=(A,C)$ be a constraint-based process model with activity set A and constraint set C. Then: A *process instance* $I=(S,\sigma)$ on S is defined by S and a corresponding trace σ.

A running process instance I is in state *satisfied* if its current partial trace σ satisfies all constraints stated in C. Furthermore, an instance is in state *violated*, if the partial trace violates all constraints stated in C and there is no suffix that can be added to satisfy them. Fig. 1 includes examples of traces of satisfied and violated instances[2] for a constraint-based model.

Considering a constraint-based model and a specific related process instance, only certain activities are enabled to be executed next (cf. Definition 4).

Definition 4. Let $S=(A,C)$ be a constraint-based process model with activity set A and constraint set C, and $I=(S,\sigma)$ be a corresponding process instance with partial trace σ. Then: An activity a_i of instance I is *enabled* at time T if a_i can be started and the instance state of I is not violated afterwards; i.e., for $\sigma=\langle e_1, e_2, \ldots e_n\rangle$, we obtain $\sigma'_l = \langle e_1, e_2, \ldots e_n, start(a_i, Rj_k, T)\rangle$ afterwards and instance (S,σ') is not in state violated.

For example, for the partial trace σ_1 of Fig. 1, *B* is enabled, while *A* is not enabled due to *C*1, and *C* is not enabled due to *C*4.

Due to their flexible nature, frequently several ways to execute constraint-based process models exist. In general, given a certain partial trace, users can choose from several enabled activities which activity to execute next, which can be a quite challenging decision. The situation is further complicated by the fact that typically multiple instances get concurrently executed within a particular timeframe. In order to ensure that the execution of a business process is optimized at a global level, the decision which activity to best execute next not only depends on a single process instance, but on the whole set of instances which are executed within a particular timeframe. Therefore, recommendation support is usually needed during the execution of constraint-based models [49].

In order to suggest suitable next activities to the user, we propose the generation of optimized enactment plans for the constraint-based model by applying CP for P&S the BP activities (cf. Section 4).

2.2. Planning, scheduling and constraint programming

For generating optimized BP enactment plans optimizing the performance goals of constraint-based process models, activities to be executed have to be planned [18] and scheduled [6] by considering the declarative specification. To do this, a constraint-based approach is proposed since, as stated, CP [35] is suitable for modeling and solving P&S problems [40].

The area of scheduling [6] includes problems in which it is necessary to determine an enactment plan for a set of activities related by temporal constraints (in our context the control-flow constraints introduced in Section 2.1). Moreover, the execution of every activity requires the use of resources, hence they may compete for limited resources. In general, the objective in scheduling is to find a feasible plan which satisfies both temporal and resource constraints. Several objective functions are usually considered to be optimized, in most cases related to temporal measures (e.g., minimization of completion time), or considering the optimal use of resources.

In a wider perspective, in AI planning [18], the activities to be executed are not established a priori, hence it is necessary to select them from a set of alternatives and to establish an ordering. In most cases, the specification of planning problems includes the initial state of the world, the goal (a predicate representing a set of possible final states) that must be reached, and a set of operators (actions) which can be applied to one state in order to reach another state. Furthermore, in planning problems, usually the optimization of certain objective functions is considered.

Constraint programming (CP) [35] (cf. Fig. 2) can be used, among others, for P&S purposes [40]. In order to solve a problem through CP, it needs to be modeled as a constraint satisfaction problem (CSP) (cf. Definition 5).

Definition 5. A *CSP* $P=(V,D,R)$ is composed by a set of variables V, a domain of values D for each variable $var_i \in V$, and a set of constraints R between variables, so that each constraint represents a relationship between a subset of variables and specifies the allowed combinations of values for these variables.

[2] For the sake of clarity, only completed events for activity executions are included in the trace representation.

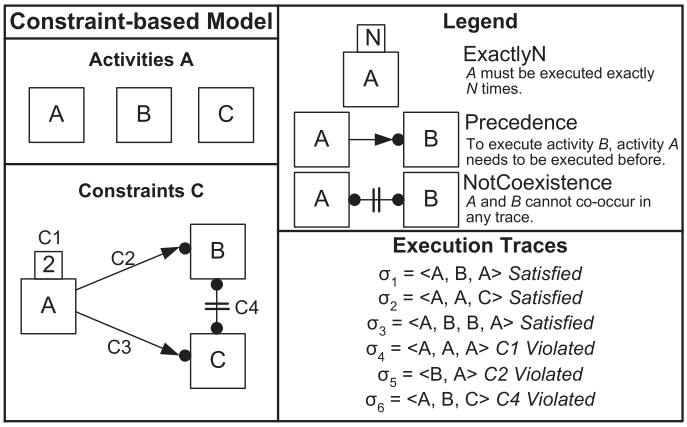


Fig. 1. Simple constraint-based model for a set of activities.

A solution (cf. Definition 6) consists of assigning values to CSP variables.

Definition 6. A *solution* $S = \langle (var_1, val_1), (var_2, val_2), \ldots (var_n, val_n) \rangle$ for a CSP $P = (V,D,R)$ is an assignment of a value val_i to each variable var_i, $\forall var_i \in V$. A solution is *partial* if there exists one or more CSP variables which are not instantiated. A solution is *feasible* when the variable-value assignments satisfy all the constraints.

In a similar way, a CSP is feasible if there exists at least one feasible solution for this CSP. From now on, S^{var} refers to the value assigned to variable *var* in the (partial) solution *S*.

Similar to CSPs, constraint optimization problems (COPs) require solutions that optimize objective functions.

Definition 7. A *COP* $P_O = (V,D,R,O)$ is a CSP which also includes an objective function *O* to be optimized.

A feasible solution *S* for a COP is *optimal* when no other feasible solution exists with a better value for the variable related to *O* (var_O).

Constraint programming allows the separation of the models from the algorithms, so that once a problem is modeled in a declarative way as a CSP, a generic or specialized constraint-based solver can be used to obtain the required solution. Furthermore, constraint-based models can be extended in a natural way, maintaining the solving methods. Several mechanisms are available for the solution of CSPs and COPs, which can be classified as *search algorithms* (i.e., for exploring the solution space to find a solution or to prove that none exists) or *consistency algorithms* (i.e., filtering rules for removing inconsistent values from the domain of the variables). In turn, search algorithms can be classified as *complete search algorithms* (i.e., performing a complete exploration of a search space which is based on all possible combinations of assignments of values to the CSP variables), and *incomplete search algorithms* (i.e., performing an incomplete exploration of the search space by repairing infeasible complete assignments or by trying to improve the objective value, so that, in general, to get a feasible or an optimal solution is not guaranteed). Various incomplete search algorithms have demonstrated their ability to obtain good solutions to difficult combinatorial optimization problems. One of the most promising of such techniques is the greedy randomized adaptive search procedure (GRASP) [13,14]. Furthermore, there exist several *hybrid search algorithms* which combine both complete and local search techniques. Large neighborhood search (LNS) [31] is a powerful hybrid technique which has recently shown outstanding results in optimizing various scheduling problems [31].

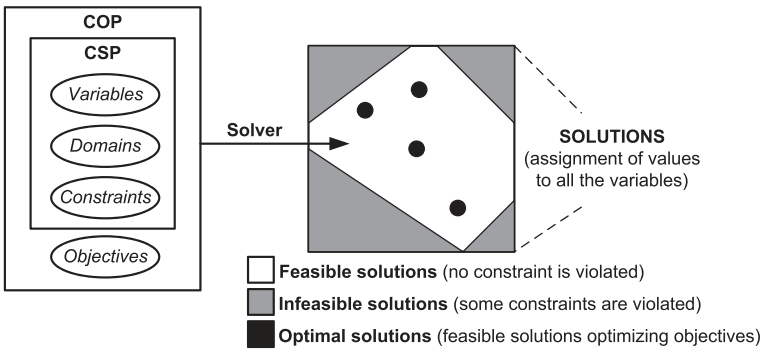


Fig. 2. Schema of constraint programming.

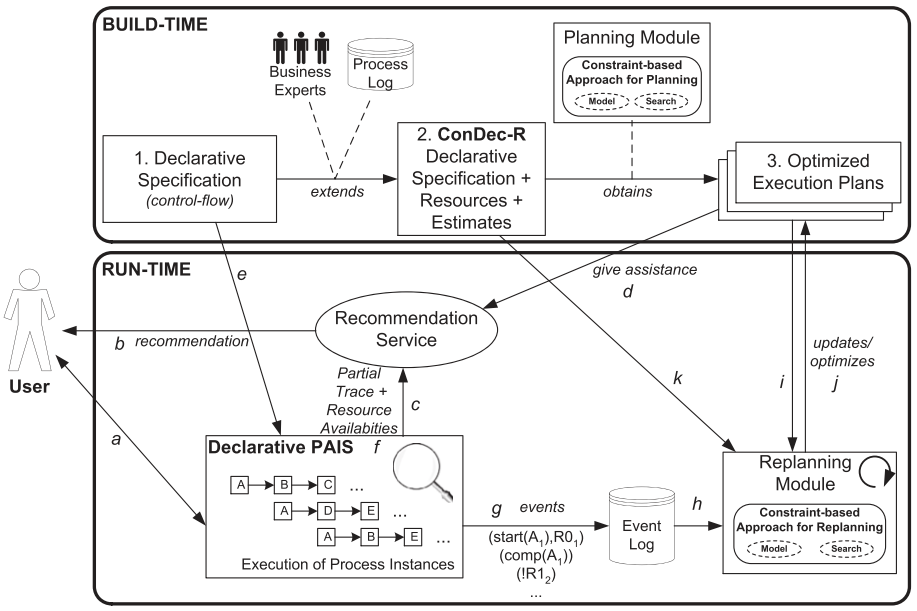


Fig. 3. Overview of our proposal for generating optimized execution plans at build-time and generating recommendations at run-time.

3. Method for generating recommendations

As stated, constraint-based processes offer much flexibility. Typically, given a constraint-based process model and a certain partial trace, users can choose from several enabled activities which activity to execute next, which is a challenging selection in most cases. In order to address this challenge we propose an approach to assist users during process execution in optimizing performance goals like minimizing the overall completion time (OCT). Specifically, users of flexible PAISs are supported during process execution by a recommendation service which provides recommendations on how to proceed best with the execution. Hereby, a recommendation (cf. Definition 8) is composed by one or more enabled activities (cf. Definition 4) to be executed next, together with their resource allocations since both control-flow and resource perspectives are considered.

Definition 8. A *recommendation Rec* is composed by a set of pairs (a_i, Rj_k) suggesting to start the i-th execution of activity a using resource Rj_k.[3]

For example, the recommendation $\langle(A_1, R0_1),(B_2, R1_2)\rangle$ suggests to start the first execution of activity A using resource $R0_1$ and the second execution of activity B using resource $R1_2$.

The recommendation service is based on optimized enactment plans which are already generated during build-time (cf. Section 3.1) by P&S all BP activities and further optimized during run-time. At specific times of the process execution, the recommendation system generates the recommendations by considering: (1) the optimized enactment plans, (2) the partial traces (cf. Definition 2) of the process instances to be optimized, and (3) the resource availabilities (cf. Section 3.2). In order to determine the recommendations, different strategies can be used (which will be described in Section 4.3). Thereby, the recommendation service ensures that not only single process instances get optimized, but the whole set of instances which is planned to be executed within a certain timeframe, hence allowing for a global optimization.

3.1. Build-time: generation of optimized execution plans

The generation of optimized execution plans comprises 3 steps (cf. Fig. 3):

(1) *Create declarative specification*. In a first step, a constraint-based model (cf. Definition 1) of the BP to be supported is created covering the control-flow through the ConDec language [46] (cf. Section 2.1). An example is depicted in Fig. 5(1), which is fully detailed in Section 5.

(2) *Extend declarative specification (including estimates and resource perspective)*. In order to plan and schedule the process activities and to cover the resource perspective, our proposal extends the constraint-based specification (cf. Definition 1) by considering resource requirements for each BP activity execution and estimates for activity durations, number of process instances executed per planning period, and resource availabilities (cf. Definition 9). Estimates can be obtained by

[3] Rj_k refers to the k-th resource with role j.

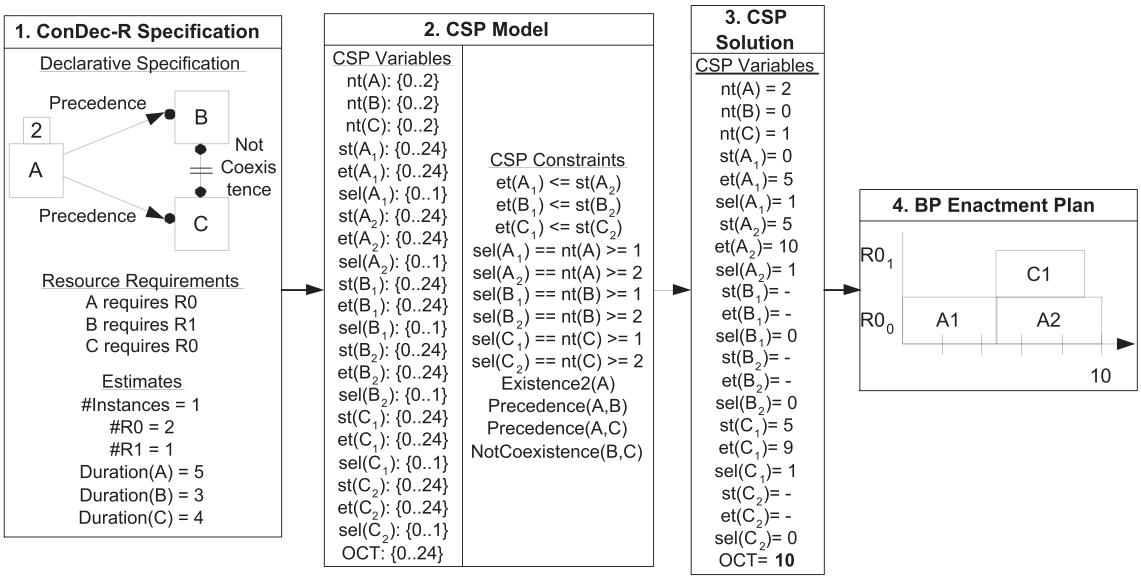


Fig. 4. Generating BP enactment plans from ConDec-R specifications through constraint programming.

interviewing business experts or by analyzing past process executions (e.g., by calculating the average values of the parameters to be estimated from event logs). Moreover, both approaches can be combined to get more reliable estimates.

Definition 9. A *ConDec-R process model* (extended constraint-based process model) $CR = (Acts, C, Res, NInst)$ related to a constraint-based process model $S = (A, C)$ is composed by a set of BP activities *Acts* which includes the estimated duration and the role of the required resource for each BP activity $a \in A$; a set of ConDec constraints *C*; a set of available resources *Res*; and the estimated number of process instances *NInst*.

In Fig. 5(1) and (2), an example of an extended constraint-based (ConDec-R) specification is depicted, which is fully detailed in Section 5.

(3) *Generate Optimized Enactment Plans*. Based on the extended constraint-based specification (cf. Definition 9) optimized enactment plans (cf. Definition 10) are generated for an estimated number of process instances (*NInst*) by applying AI techniques for P&S the BP activities (cf. Fig. 4). We propose an efficient and innovative approach for P&S the BP activities to get BP optimized plans (cf. Section 4).

Definition 10. A *BP enactment plan* EP is composed by: (i) the number of times each BP activity is executed, (ii) the start and the completion times for each activity execution, and (iii) the resource which is used for each activity execution.

The enactment plans can be graphically represented by a Gantt chart [16], which illustrates the start and completion times of each activity execution together with the resource allocation. In Fig. 5(3), an example of an optimized BP enactment plan is depicted, which is fully detailed in Section 5.

Since the generation of optimal plans presents NP-complexity [17], it is not possible to ensure the optimality of the generated plans for all cases. The developed constraint-based approach (cf. Section 4), however, allows solving the considered problems in an efficient way, i.e., plans of high quality (cf. Definition 11) are generated, as demonstrated in Section 6.

Definition 11. The *quality of a BP enactment plan* EP (cf. Definition 10) related to a ConDec-R process model CR (cf. Definition 9) is defined by $EP_*{}^{OCT}/EP^{OCT}$, where EP^{OCT} is the value of *OCT* in the plan *EP*, and $EP_*{}^{OCT}$ is the optimal value of *OCT* for *CR*.[4]

3.2. Run-time: generating recommendations on possible next execution steps

This section describes how the optimized plans are then used for assisting users during process execution. At run-time, process instances (cf. Definition 3) are executed by authorized users (a in Fig. 3). At any point during the execution of a process instance, the user can select from the set of enabled activities (cf. Definition 4) what to do next. However, to guide the user to optimize the overall process goals, recommendations (cf. Definition 8) are provided by the recommendation service (b in Fig. 3). Note that the

[4] Since the generation of optimal plans presents NP complexity (and hence, the optimal plan can be unknown), the quality value is approximated by EP'^{OCT}/EP^{OCT}, where EP'^{OCT} is the value for *OCT* in the best feasible plan which is known for CR when the optimum is unknown.

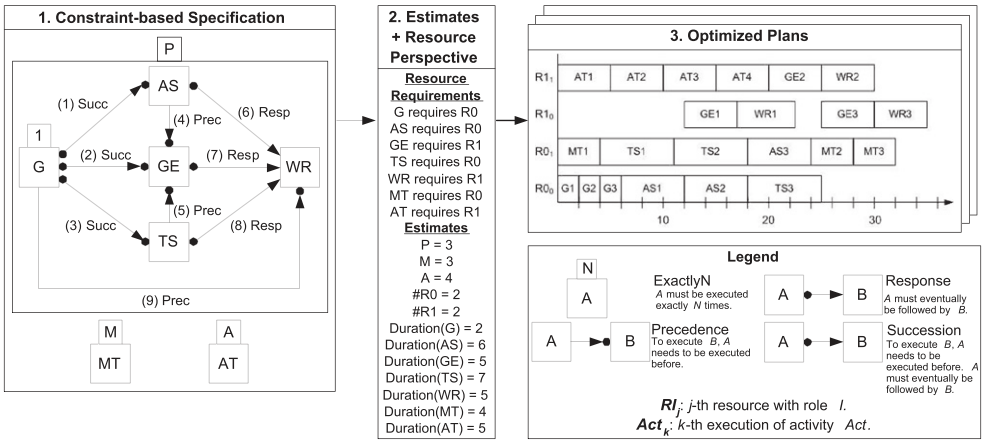


Fig. 5. Running example (built-time perspective): generating optimized execution plans from a ConDec-R specification.

user is not obliged to follow the recommendations but she can select any of the enabled activities, i.e., all the flexibility of the declarative specification is kept. To provide recommendations, the recommendation service proposes the most suitable activity to execute next, i.e., proposes the recommendation with the highest quality (cf. Definition 12). Note that the quality of the recommendations which are generated is determined by the quality of the BP enactment plans (cf. Definition 11) which are used.[5]

Definition 12. The *quality of a recommendation Rec* (cf. Definition 8) related to a ConDec-R process model CR (cf. Definition 9) is equal to the quality of the BP enactment plan (cf. Definition 11) which was used for generating *Rec*.

Algorithm 1 shows how the recommendations are generated. As input data some information is required: (i) the ConDec-R specification of the problem (cf. Definition 9) and (ii) the initial optimized enactment plans (cf. Definition 10) generated during the build-time phase. As stated, for a particular timeframe a BP enactment plan for a set of instances (cf. Definition 3) is generated. Algorithm 1 starts at the beginning of such a timeframe and lasts until all the planned instances have completed (line 15 in Algorithm 1).

Algorithm 1. Provide recommendations

```
   input : ConDec-R Specification cr
           set<EnactmentPlan> plans
1  Recommendation rec;
2  Set<Event> allEvents ← ∅;
3  Set<Event> newEvents;
4  int T ← currentTime();
5  repeat
6      if event(newEvents, T) then
7          allEvents ← allEvents ∪ newEvents;
8          plan ← update(cr, plans, allEvents);
9      if optimizerPlan(cr, plans, allEvents)! = null then
10         plan ← optimizerPlan(cr, plans, allEvents);
11     rec ← generateRecommendation(plans, allEvents);
12     if rec! = null then
13         send(rec);
14     T ← currentTime();
15 until !CompleteTrace(cr, allEvents) ;
```

Algorithm 1 continuously generates recommendations (line 11) on how to proceed with process execution considering (1) the best available enactment plan (d in Fig. 3) meeting the constraints imposed by the constraint-based specification (e in Fig. 3), and (2) all events that occurred during process execution (i.e., *allEvents*). This includes (1) the current partial traces of the process instances (c in Fig. 3), and (2) the current information about resource availabilities (c in Fig. 3), e.g., $(!Rj_k, T)$ means that k-th resource with role j becomes unavailable at time T (cf. Fig. 6). In the case that a recommendation is suggested (line 12), the recommendation system sends it to the user (line 13).

As execution proceeds, the BP enactment and the resource availabilities are monitored (f in Fig. 3). If there are new events at time T (line 6 in Algorithm 1), i.e., activities get started/completed or resources become available/unavailable (g in Fig. 3), then

[5] For the current work, the durations of the recommendation generation is considered negligible compared to the duration of the process activities.

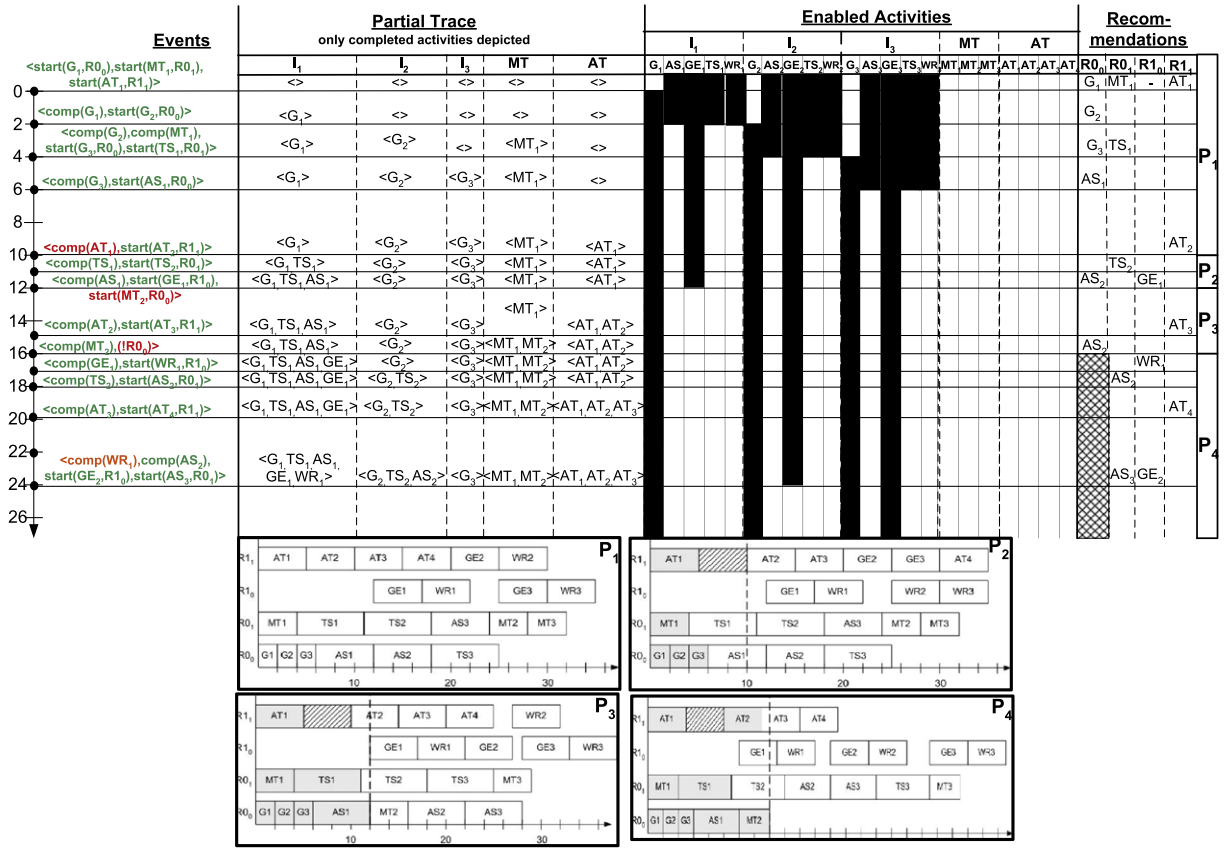


Fig. 6. Running example (run-time perspective): giving recommendations on possible next steps.

the set of events *allEvents*, which includes both the partial trace and the resource availability events, is updated (line 7 in Algorithm 1). By doing this, the proposed approach is able to deal with the uncertainty involved (i.e., inaccurate estimates, unexpected changes in resource availabilities, and user deviations).

Whenever events are updated the replanning module (h in Fig. 3) analyzes the optimized plans (i in Fig. 3) as well as the events. In particular, it checks if the current execution traces match with any of the optimized enactment plans (and if a recommendation can be made) or whether updates of the execution plans are needed (j in Fig. 3). In general, updates of the execution plan can become necessary due to deviations (line 8 in Algorithm 1), i.e., (i) the execution trace is not part of one of the optimized enactment plans (e.g., the user is not always following the recommendations), (ii) estimates are incorrect (e.g., when activity executions take longer/shorter than estimated, or more or less instances than expected get executed), or (iii) resource availabilities change (i.e., resources become unavailable). Note that not every deviation requires replanning (some examples are given in Section 5).

Moreover, the replanning module is continuously searching for a better plan by considering the event log during BP execution, provided that the current plan is not optimal. In this way, plan updates are conducted whenever the replanning module finds a solution which is better than the current optimized plans (lines 9 and 10 in Algorithm 1). If plan updates are required, the replanning module needs to access the extended constraint-based specification of the process (k in Fig. 3) to generate new optimized plans considering both the estimates and the constraint-based specification. If necessary, the replanning, i.e., the generation of new optimized enactment plans, is carried out by applying a constraint-based approach for P&S the BP activities (cf. Section 4).

Despite the NP-complexity of the considered problems, replanning is usually less time consuming than initial planning, since most of the information about previous generated plans can usually be reused, and CSP variable values become known as execution proceeds (cf. Section 6).

A running example illustrating the complete process is detailed in Section 5.

4. Method for generating optimized BP enactment plans

As stated during build-time optimized execution plans are generated.[6] In this work, CP is selected for the generation of the optimized plans since it supplies a suitable framework for modeling and solving problems involving P&S [40]. Moreover, CP allows the

[6] A proof-of-concept prototype has been implemented through a web-based application, which allows for the generation of optimized enactment plans from constraint-based specifications. It can be accessed at http://regula.lsi.us.es/OptBPPlannerV2/.

separation of the models from the algorithms, so that once a problem is modeled as a CSP, a generic or specialized constraint-based solver can be used to obtain the required solution.

In this work, we present a new and efficient constraint-based approach for P&S the BP activities specified through the ConDec-R language (i.e., ConDec specification, resource requirements and estimates) optimizing process performance goals. To improve the modeling of the problems (cf. Section 4.1) and to efficiently handle the constraints in the search for solutions, our constraint-based proposal includes for each ConDec template a filtering rule (responsible for removing values which do not belong to any solution) (cf. Section 4.2). Moreover, efficient search algorithms for solving the CSP are described in Section 4.3.

Fig. 4 shows the complete process which is proposed to generate BP enactment plans from a ConDec-R specification through CP. Step 1 consists of the specification of the BP model through ConDec-R, i.e., including the constraint-based specification, the resource requirements and the estimates. This ConDec-R specification is then translated into a CSP (cf. Step 2) based on which optimized enactment plans are obtained using different search algorithms (cf. Step 3), and visualized as a Gantt chart (cf. Step 4).

4.1. Representing the constraint-based model as CSP model

As stated, BP activities and constraints are specified in a ConDec-R model so that frequently several feasible ways to execute this model exist. Each specific feasible execution of a ConDec-R model leads to a specific value for the function to optimize, i.e., the overall completion time of the process instances. In general, multiple optimal executions, i.e., feasible solutions leading to a minimal completion time, may exist. In order to generate optimal (or optimized) execution plans (cf. Definition 10) for a specific ConDec-R model, we propose a constraint-based approach for P&S the BP activities.

The first step for solving a problem through CP consists of modeling the problem as a CSP. Regarding the CSP model of the proposed constraint-based approach, BP activities (repeated activities, cf. Definition 13), which can be executed arbitrarily often if not restricted by any constraints, are modeled as a sequence of optional scheduling activities (cf. Definition 14), since each execution of a BP activity is considered as one single activity which needs to be allocated to a specific resource and temporarily placed in the enactment plan, i.e., stating values for its start and end times.

Definition 13. A *repeated activity a* is a BP activity which can be executed several times, i.e., several instances of the same activity can exist for one process instance.

Definition 14. A *scheduling activity* (or each repeated enactment of an activity) a_i represents the i-th execution of a repeated activity a, i.e., a specific BP activity instance.

In this way, there are two main types in the proposed CSP model: (1) a type representing the BP activities, named *RepeatedActivity*, and (2) a type representing each execution of the BP activity, named *SchedulingActivity*. The properties of *RepeatedActivity* (cf. Definition 9) are: (i) *duration*: represents the estimated duration of the BP activity (the same estimated duration is considered for all the executions of a BP activity), (ii) *role*: indicates the role of the required resource for the activity execution (the same required resource is considered for all the executions of a BP activity), and (iii) *nt*: showing the actual number of times the BP activity is executed. In ConDec-R, *nt* can be unknown or have a value (if the number of activity executions is restricted in the ConDec model through existence constraints), and hence *nt* is a CSP variable which needs to be instantiated during the CSP solving process. There are lower and upper bounds for *nt*, as depicted in Fig. 4 (*LB*(*var*) and *UB*(*var*) refer to the lower and upper bounds of the domain of CSP variable *var*, respectively). The *RepeatedActivity* type is composed of *nt SchedulingActivities*. Each *SchedulingActivity* includes: (i) *st*: indicating the start time of the activity execution, (ii) *et*: indicating the end time of the activity execution (each execution of a BP activity needs to be temporarily placed in the enactment plan), and (iii) *sel*: indicating whether or not the *SchedulingActivity* is selected to be executed. As stated, in ConDec-R, the number of times each BP activity is to be executed can be unknown (*nt* variable), and hence each associated scheduling activity can potentially be executed (i.e., included in the enactment plan), which is modeled through the *sel* variable, i.e., *sel* is equal to 0 if it is not executed, 1 otherwise.

Henceforth, $st(a_i)$ and $et(a_i)$ represent the start and the end times of activity a_i, respectively.

In our approach, considering the minimization of the overall completion time of the process enactment, a CSP variable for this function, named *OCT*, is also included in the CSP model: $OCT = \max_{a \in A}(et(a_{nt(a)}))$.

Definition 15. A *CSP-ConDec problem* related to a ConDec-R process model $CR = (Acts, C, Res, NInst)$ (cf. Definition 9) is a COP $P_O = (V, D, R, O)$ (cf. Definition 7) where:

1. The set of variables V is composed by all the CSP variables included in the presented CSP model plus the CSP variable related to the overall completion time (*OCT*), i.e., $V = \{nt(a), \forall a \in Acts\} \cup \{st(a_i), et(a_i), sel(a_i), \forall i \in [1 \dots UB(nt(a))], \forall a \in Acts\} \cup OCT$.
2. The set R is composed by the global constraints (implemented by the filtering rules, cf. Section 4.2) related to the ConDec-R constraints included in C together with the constraints from the proposed CSP model, i.e., $\forall i$: $1 \le i < nt(a)$: $et(a_i) \le st(a_{i+1})$, $\forall i: 1 \le i \le nt(a)$: $sel(a_i) = 1$ and $\forall i > nt(a): sel(a_i) = 0$ for each repeated activity $a \in Acts$.
3. The set of domains D is composed by the domains for each variable included in V.
4. The objective function O is minimizing the *OCT* variable.

In the proposed constraint-based approach resources are implicitly constrained since most constraint-based systems provide a high-level constraint modeling specific to scheduling which includes an efficient management of shared resources. Note that,

besides the role-based allocation pattern, the CSP variables which are included in the model can be also used for specifying further resource constraints [38].

Fig. 4 includes the translation from a ConDec-R specification into a CSP so that the CSP variables and constraints are stated as explained in Definition 15 (cf. step 2). For all the activities, the value for $LB(nt(act))$ is initially set at 0 (it will be automatically updated if existence constraint is added), and the value for $UB(nt(act))$ is set at the maximum cardinality of activities, i.e., 2 for the constraint-based model depicted in Fig. 4, since this value states the minimum *nt* for all the BP activities which ensures a feasible solution (the optimized solution, in general, includes lower values of *nt* for several activities). Moreover, Fig. 4 depicts upper and lower bounds for *OCT*. Thereby, $LB(OCT)$ is initially set at 0, and $UB(OCT)$ is set at the maximum cardinality times the sum of the duration of all the BP activities, i.e., $2\times(5+3+4)$ in the example, since the worst solution which can be obtained results in a plan which includes the execution of each BP activity the maximum number of times (at least *nt* for one activity must be established, otherwise any activity will be included in the plan), and without considering parallelism in the execution of any activity.

4.2. Filtering rules

The proposed constraint-based approach includes specific filtering rules (i.e., responsible for removing values which do not belong to any solution from the domains of variables) for the definition of the high-level relationships between the BP activities, so that the constraints stated in the ConDec-R specification (cf. Definition 9) are included in the CSP model through the related filtering rules. These filtering rules facilitate the specification of the problem through global constraints at the same time as they enable the efficiency in the search for solutions to increase. The developed filtering rules (cf. [3]) are considered in all the proposed search algorithms.

4.3. Search algorithms

The proposed approach is used for the generation of optimized enactment plans during both build-time (cf. Section 3.1) and run-time (cf. Section 3.2).

Once a CSP is modeled, several constraint-based mechanisms can be used to obtain the required solution. In this section, some search algorithms which efficiently deal with CSP-ConDec problems are introduced.

In the current work, we adapt and apply existing methods, specifically complete search [35], incomplete search [35], and GRASP [13,14] (cf. Section 2.2), for solving the specific considered problems and we also evaluate their suitability for the generation of recommendations (cf. Section 6).

In general, when optimizing a CSP variable, if a feasible solution which is known exists, the value of the variable to optimize in the known solution can be used for discarding large subsets of fruitless candidates by using upper and lower estimated bounds of the quantity being optimized during the search process. Thus, if a known feasible solution *S* for the problem to solve exists, the objective value for this solution (S^{OCT}) is a valuable information which can be added to the constraint model through the constraint $OCT<S^{OCT}$. Thus, some non optimal candidates, i.e., candidates whose *OCT* value cannot be less than S^{OCT} in any case, are discarded during the search, and hence increases the efficiency in the search for solutions.

Moreover, in our proposal, during the search process, some of the values which only lead to non-feasible solutions, i.e., inconsistent values, are removed from the domains of the CSP variables in order to reduce the search space through maintaining arc consistency (cf. Definition 16).

Definition 16. A $CSP=(V,D,R)$ presents *arc consistency* if for all pairs of CSP variables $(var_1, var_2)|var_1, var_2\in V$, for each value of var_1 in the domain of var_1 there is some value in the domain of var_2 that satisfies all the constraints stated in *R* between var_1 and var_2, and vice versa.

In the proposed approach, the developed filtering rules (cf. Section 4.2) and CSP modeling (cf. Section 4.1) are implemented such that they maintain the arc consistency for all pairs of CSP variables during all the search process.

Proposition 1. *Let S be the best complete solution, i.e., with fewest overall completion time, which can be obtained for a CSP-ConDec problem P by considering certain fixed values for all nt variables* $(nt_{Act_1}, nt_{Act_2}, \ldots, nt_{Act_{Act}})$, *i.e.,* $S^{nt(Act_i)} = nt_{Act_i}, \forall i\in\{1\ldots Act\}$. *Let S′ be the best complete solution which can be obtained for P by considering certain fixed values for all nt variables* $(nt_{Act_1}, nt_{Act_2}, \ldots, nt'_{Act_i}, \ldots, nt_{Act_{Act}}, \quad nt'_{Act_i} = nt_{Act_i} + 1)$. *Then:* $S'^{OCT}<S^{OCT}$ *is not possible.*

Proof. Let $P_O=(V,D,R,O)$ be a CSP-ConDec problem (cf. Definition 15) related to a ConDec-R process model $CR=(Acts,C,Res,NInst)$ (cf. Definition 9). Increasing the number of times a repeated activity is executed has different effects depending on the kind of high-level constraints stated in *C*:

- Case 1: $\nexists c\in C$ of type *Alternate* or *Chain*, i.e., including disjunctions related to existential forms since these relationships imply that between each two executions of a specific BP activity, at least one execution of another specific BP activity must exist (cf. [3]). In this case, the CSP which is obtained after instantiating all the *nt* variables can be represented by a precedence graph, i.e., an acyclic directed graph where nodes correspond to activities and there is an arc from *A* to *B* if *A* must precede *B*. In this way, the fact of increasing the value of any *nt* variable results in including one additional scheduling activity in the previous precedence graph. Therefore, the new CSP, resulted from increasing one of the *nt* variables (i.e., adding a new scheduling activity), can be represented by a precedence graph which extends the previous graph by including the precedence constraints in which the new scheduling

activity is involved. Taking into account that $OCT = \max(et(Act_{nt(Act)}))$, $\forall Act \in A$, increasing the number of times a repeated activity is executed does not make improving the optimal solution possible.

- Case 2: $\exists c \in C$ of type *Alternate* or *Chain*, i.e., some disjunctions related to the existential forms of alternate and chain templates (cf. [3]) exist. These disjunctions can result in having a set of possible alternative precedence graphs *PGs*, so that one of the graphs included in the set *PGs* leads to the optimal solution of the problem. The fact of increasing one of the *nt* variables results in adding a new scheduling activity to the precedence graph (together with the precedence constraints in which this activity is involved in). Moreover, the existential relationships can be modified due to adding this new activity. The fact of adding all these new relationships between activities implies that each graph which belongs to the new set of precedence graphs *PGs′* corresponds to a reinforcement of some of the original graphs which belonged to *PGs*, and hence any graph belonging to *PGs′* can lead to a solution with less *OCT* value.

As stated, the arc consistency for all pairs of CSP variables is maintained during entire the search process. In our proposal, after posting all ConDec relationships between the BP activities, i.e., adding the arc consistent filtering rules, all the *nt* variables are instantiated to *LB*(*nt*). If the resulting CSP is feasible, then the optimal solution for this CSP is also an optimal solution for the original CSP-ConDec problem (Proposition 1). Otherwise, when the resulting CSP is unfeasible, the values of the *nt* variables are increased step by step. In this way, the optimal solution is searched by considering the CSP which is obtained as a result of instantiating *nt* variables in the first feasible solution which is found, i.e., for the fewest feasible values of *nt*. This optimal solution is also the optimum for the original CSP-ConDec problem (Proposition 1). Usually, the instantiation of all *nt* variables to *LB*(*nt*) is feasible due to the arc consistency which is maintained. However, this may not be true for some combinations of *Alternate* or *Chain* relationships, and hence, greater values for *nt* variables need to be considered.

After instantiating all the *nt* variables to a fixed value, the search for the optimal solution only entails the consideration of the remaining CSP variables.

4.3.1. Complete search

As stated, complete search consists of exploring a search tree for the CSP problem which is based on all possible combinations of assignments of values to the CSP variables. In general, both the ordering of instantiation of the variables and the ordering of selection of values for each variable have a great influence on the efficiency of the search process and also on the quality of the solutions which are obtained.

Once the *nt* variables of the repeated activities are instantiated, the considered problem becomes an extension of the cumulative job shop scheduling problem [1], CJSSP. While CJSSP considers sequences of activities related by precedence constraints which require some shared discrete resources and which must be scheduled in order to minimize some objectives, our extension includes further kinds of relationships, e.g., alternate or chain [46] which are not pure precedence relationships considered in typical scheduling problems. In our proposal for performing a complete search, after generating a first feasible solution by using Algorithm 5 (detailed later), we use an efficient method for solving the CJSSPs, named *setTimes* [22], based on [50] (cf. [10]).

4.3.2. Iterative bounded greedy

Due to the NP-complexity of the considered problem, in addition to the complete search, an incomplete search approach, named iterative bounded greedy (IBG), is implemented including randomized components to achieve diversified results (cf. Algorithm 2). In Algorithm 2, a greedy randomized algorithm (Algorithm 5) is used (line 3) for iteratively improving the best solution found (line 4), until a time limit is reached. The best solution over all iterations is returned as the result (line 5).

Algorithm 2. Iterative bounded greedy

```
   input  : Set<RepeatedActivities> repAct
            Set <Template> templates
   output: Solution bestSol
1  Solution sol;
2  while Iterative Bounded Greedy stopping criterion not satisfied do
3  |  sol ← ConstructGreedyRandomizedSolution(repAct, templates);
4  |  UpdateSolution(sol, bestSol);
5  return bestSol;
```

With the proposed incomplete search, all the solutions can be reached and the search procedure efficiently explores a wide range of solutions from diversified areas of the search space.

4.3.3. GRASP-LNS

In addition to complete and incomplete search, a hybrid approach is considered. GRASP [13,14] (cf. Algorithm 4) consists on an iterative process in which each iteration includes two phases: (i) a construction phase (line 3), in which a feasible solution is built through Algorithm 5, and (ii) a local search phase, i.e., incomplete search (line 4), in which the neighborhood of the generated solution is explored to find a local optimum. The best solution over all GRASP iterations (line 5) is returned as the result (line 6).

Algorithm 3. GRASP-LNS

```
input  : Set<RepeatedActivities> repAct
         Set <Template> templates
output: Solution bestSol
1 Solution sol;
2 while GRASP-LNS stopping criterion not satisfied do
3 |   sol ← ConstructGreedyRandomizedSolution(repAct, templates);
4 |   LocalSearch(sol);
5 |   UpdateSolution(sol, bestSol);
6 return bestSol;
```

In the current approach, LNS [31] is used for exploring the neighborhood of current solutions in the GRASP algorithm (line 4 of Algorithm 4), resulting in an efficient hybrid technique, named GRASP-LNS. In a LNS algorithm (cf. Algorithm 4), in each iteration, a neighborhood is explored with CP trying to improve the current best solution (*bestSol* variable) in the following way: first, part of the current solution is relaxed (line 5 of Algorithm 4) so that the domain of some variables is restored to its initial range, while fixing the remaining variables to their current value; secondly, the restricted problem is re-optimized by using CP with a limit on the number of failures (line 6 of Algorithm 4). The best solution over all iterations (line 7 of Algorithm 4) is returned as the result (line 8 of Algorithm 4). In Algorithm 4 the reason for setting a failure limit is to avoid exploring a neighborhood for too long, allowing the search to explore a variety of neighborhoods.

Algorithm 4. LocalSearch

```
input  : Solution sol
output: Solution bestSol
1 PartialSolution pSol;
2 Solution tempSol;
3 bestSol ← sol;
4 while bestSol can be improved AND a failure limit does not occur do
5 |   pSol ← Relax(bestSol);
6 |   tempSol ← Re − optimizeCP(pSol);
7 |   UpdateSolution(tempSol, bestSol);
8 return bestSol;
```

4.3.4. Greedy generation of a feasible solution

As follows, a greedy randomized algorithm which is used for the generation of feasible solutions is presented. This algorithm is invoked by the different proposed searches, i.e., complete search, iterative bounded greedy, and GRASP-LNS, as explained before. After instantiating all *nt* variables for all the repeated activities, a feasible solution can be quickly generated by Algorithm 5.

Algorithm 5. ConstructGreedyRandomizedSolution

```
input  : Set<RepeatedActivities> repAct
         Set<Template> templates
output: Solution sol
1  Set<SchedAct> allowedActs ← ∅;
2  Set<SchedAct> actAlreadyInstantiated ← ∅;
3  Map<RepeatedActivities,Integer> instantiatedN(repAct) ← {0, . . . , 0};
4  repeat
5  |   allowedActs ← ∅;
6  |   foreach A in repAct do
7  |   |   int nextA ← instantiatedN(A) + 1;
8  |   |   if nextA ≤ nt(A) then
9  |   |   |   if Allow(A, nextA, templates, actAlreadyInstantiated) then
10 |   |   |   |   allowedActs ← allowedActs ∪ A_nextA;
11 |   if allowedActs ≠ ∅ then
12 |   |   SchedAct actToInst ← Select(allowedActs);
13 |   |   Instantiate(actToInst, sol);
14 |   |   actAlreadyInstantiated ← actAlreadyInstantiated ∪ actToInst;
15 |   |   instantiatedN(actToInst) = instantiatedN(actToInst) + 1;
16 until allowedActs == ∅ ;
17 return sol;
```

The main idea of Algorithm 5 consists of instantiating the value of certain allowed variables related to a specific P&S activity in each step, i.e., those variables which can be instantiated by taking the P&S activities which have been already instantiated and the set of relationships (templates) into account. In lines 1 and 2, both the set of activities allowed to be instantiated in the next step and the set of activities previously instantiated are created and initialized. Furthermore, a map which relates each repeated activity A to the last execution of A which has already been instantiated is created and initialized to 0 (line 3). In each step, the set *allowedActs* is filled with the P&S activities which are allowed to be instantiated next (lines 5–10), by considering that only one execution of each repeated activity is analyzed to be included since the P&S activity related to the i-th execution of A can only be instantiated after instantiating the P&S activity related to (i-1)-th execution of A. In this way, for each repeated activity A (line 6), the index of the P&S activity related to the execution of A to be instantiated next is stored in the variable *nextA* (line 7). If there is any execution of A which remains to be executed (line 8), then the method *Allow* checks if this activity instance can be instantiated next (line 9), i.e., is enabled (cf. Definition 4). In the affirmative case, the related P&S activity is included in the set *allowedActs* (line 10). If there is any activity allowed to be instantiated next (line 11), one of these activities is selected (line 12). After the selection of the P&S activity to be instantiated next, the start and the end variables of this selected activity are instantiated to the minimum value of its domains (line 13). These instantiations, in general, result in updating the domain of some CSP variables (the developed filtering rules are in charge of carrying these updates out). Moreover, the instantiated activity is included in the set *actAlreadyInstantiated* (line 14), and its related information is updated (line 15). The lines 5–15 are repeated until a solution is completely constructed (line 16). The generated solution is returned as the result (line 17).

A feasible good solution can be swiftly generated through Algorithm 5 by considering different heuristics for the implementation of the *Select* method (line 11 of Algorithm 5). In this proposal, for the IBG (cf. Algorithm 2) and GRASP-LNS (cf. Algorithm 3) searches, in order to get diversified results each time Algorithm 5 is invoked, the *Select* method is implemented (heuristic) so that the activities are selected by considering the probability $\pi(A)$ of selecting an activity A as:

$$\pi(A) = \frac{1/r(A)}{\sum_{B \in Acts} 1/r(B)}$$

where $r(A)$ denotes the rank of A when all the candidates are ranked according to their earliest start time.

On the other hand, for the complete search, the heuristic which is considered selects the activity with the lowest starting time, i.e., the most promising one, since for the considered complete search Algorithm 5 is only used for generating an initial solution. In most cases, this initial solution will be improved as the complete search proceeds.

Notice that for a feasible combination of *nt* values, Algorithm 5 always generates a feasible solution, since the arc consistency is maintained during all the process by the developed filtering rules.

5. A running example

In this section, the proposed approach is used for managing recommendations during a hypothetical execution of a running example which represents a travel agency. This agency manages holiday bookings by offering clients the following three services: transport, accommodation, and guided excursions. After the client request is carried out, the agency must write a report which contains the information in answer to the request, which will then be sent to the client. Besides managing the client requests, people who work in the agency must perform further activities related to management, and accounting tasks. The number of client requests (P), management tasks (M) and accounting tasks (A) which must be dealt with during a working day is known at the beginning of the day, hence it is necessary to organize the work considering the estimated work load. The objective to be considered by the agency is to minimize the overall completion time of the daily processes. However, this example can easily be extended in order to consider the optimization of further objective functions, such as cost. In the travel agency, optimized BP enactment plans must be created every day to generate recommendations about the activities to be executed and the correct ordering. The activities which must be executed to deal with the client requests, management and accounting tasks are detailed in Table 1.

For activities which are executed more than once, each execution must finish before the next execution can start. Moreover, in order to simplify the analyzed problem, we assume that all the executions of the same activity require a resource of the same role with the same duration.[7]

5.1. Build-time phase

To solve this problem through the proposed approach, the first step is the creation of the related ConDec specification. The constraint-based specification includes seven activities, G, AS, GE, TS, WR, MT and AT, and several relationships (ConDec templates [46]) between the activities (cf. Fig. 5(1)): (i) after a client request (G), transport and accommodation search (TS and AS), and guided excursion elaboration (GE), must be processed, and before the execution of these services (AS, TS and GE), the client request (GR) must be received (Relationships (1), (2) and (3) in Fig. 5(1)); (ii) before preparing the guided excursion (GE), accommodation and transport must be known (Relationships (4) and (5) in Fig. 5(1)); (iii) any execution of activities related to

[7] Note that the proposed approach can deal with BP activities which require several resources of various kinds of roles, since the considered problems are modeled as scheduling problems with optional activities, where the resources have a discrete capacity.

Table 1
BP activities to deal with the client requests, management and accounting tasks.

ID	Description	Role	Duration
G	The client request is received	$R0$	2
AS	A suitable accommodation is searched	$R0$	6
GE	Guided excursions are organized	$R1$	5
TS	A suitable transportation is searched	$R0$	7
WR	A report with the answers to the requests is written	$R1$	5
MT	Management task is carried out	$R0$	4
AT	Accounting task is carried out	$R1$	5

client requests, i.e., AS, TS and GE, must be reported (Relationships (6), (7) and (8) in Fig. 5(1)); and (iv) the report cannot be written before a client request (Relationship (9) in Fig. 5(1)).

In a next step, the constraint-based specification is extended with resource requirements, estimates for the number of instances to be executed, resource availabilities, and the duration of the activities (cf. Fig. 5(2)). Lastly, the constraint-based approach is applied to generate optimized enactment plans for the specified problem (cf. Fig. 5(3)).

5.2. Run-time phase

Based on the execution plans generated during build-time, recommendations are then generated during run-time as explained in Section 3.2. For generating the recommendations, the optimized enactment plans are generated by selecting the best solution which is obtained by any of the three proposed search algorithms (cf. Section 4.3), i.e., complete search, iterative bounded greedy, and hybrid search (i.e., the enactment plan with the highest quality is selected, cf. Definition 11). As explained in Section 3.2 recommendations are meant to help users to find good/optimal ways to execute the process, i.e., recommendations only provide guidance, but are not enforcing any behavior and users can decide to deviate at any time. Therefore, users are allowed to do everything which is not prohibited by the constraints. The proposed approach not only considers user deviations but also other deviations such as unexpected changes in resource availabilities or inaccurate estimations. All deviations which take place during the process enactment are mimicked by events in order to be analyzed by the replanning module to check if replanning is required (cf. Fig. 3, Algorithm 1).

Fig. 6 shows the behavior of the proposed recommendation service (cf. Section 3.2) when hypothetical process instances with given traces are executed for the constraint-based specification of Fig. 5, for three client requests (P = 3), three management tasks (M = 3) and four accounting tasks (A = 4). To demonstrate how the recommendation system behaves when deviations take place, the traces of the hypothetical process instances include different types of deviations.

At the beginning of the execution, plan P_1 (which has already been generated during build-time considering the estimates for P, M and A) is considered for the generation of the recommendations. Initially, the partial trace for all instances is empty, which can be seen in the column Partial Trace, where completed events for activity executions are depicted. Furthermore, for all three client requests (i.e., I_1, I_2 and I_3), G is enabled (reflected by the white bars in columns G_1, G_2, G_3), whereas AS, GE, TS, and WR of instances I_1, I_2 and I_3 are not yet enabled (reflected by the black bars in columns AS_i, GE_i, TS_i, WR_i, $\forall i \in \{1,2,3\}$). Moreover, MT and AT are always enabled since there is no constraint restricting their execution. Activities AS, GE and TS are not enabled since G must be executed before executing AS, GE and TS due to the succession constraints. In a similar way, activity WR is not enabled since the execution of WR requires a previous execution of G. At time 0, starting execution of G_1 using $R0_0$, MT_1 using $R0_1$, and AT_1 using $R1_1$ is suggested. The user follows the recommendation. Due to `Exactly_1(G)`, G_1 is not enabled anymore. At time 2, G_1 is completed, hence AS_1, TS_1 and WR_1 becomes enabled, and the partial trace of I_1 contains G_1. Activity GE_1 is not yet enabled since the execution of GE_1 requires a previous execution of both AS_1 and TS_1. Furthermore, at time 2, starting execution of G_2 using $R0_0$ is suggested. The user follows the recommendation. Due to `Exactly_1(G)`, G_2 is not enabled anymore. At time 4, G_2 and MT_1 are completed, hence AS_2, TS_2 and WR_2 become enabled. Furthermore, at time 4, starting execution of G_3 using $R0_0$ and TS_1 using $R0_1$ is suggested. The user follows the recommendation. Due to `Exactly_1(G)`, G_3 is not enabled anymore. At time 6, G_3 is completed, hence AS_3, TS_3 and WR_3 becomes enabled. Furthermore, at time 6, starting execution of AS_3 using $R0_0$ is suggested. The user follows the recommendation. At time 10, AT_1 is completed five time units later than expected. Since there was no slack time between AT_1 and AT_2, P_1 becomes outdated, and the replanning module generates P_2 considering the new conditions.

At time 10, based on P_2 starting execution of AT_2 using $R1_1$ is suggested. The user follows the recommendation. At time 11, TS_1 is completed. Furthermore, at time 11, starting execution of TS_2 using $R0_1$ is suggested. The user follows the recommendation. At time 12, AS_1 is completed, hence GE_1 becomes enabled. Furthermore, at time 12, starting execution of AS_2 using $R0_0$ and GE_1 using $R1_0$ is suggested. The user decides to partially follow the recommendation, so that, instead of executing AS_2 she decides to start MT_2. After this unexpected decision, P_2 becomes invalid, and the replanning module generates P_3 considering the new conditions.

At time 15, AT_2 is completed. Furthermore, at time 15, based on P_3, starting execution of AT_3 using $R1_1$ is suggested. The user follows the recommendation. At time 16, MT_2 is completed. Furthermore, at time 16, an unexpected event occurs (i.e., resource $R0_0$ became unavailable), hence P_3 is no longer valid, and the replanning module generates P_4 considering the new conditions.

At time 17, GE_1 is completed. Furthermore, at time 17, based on P_4, starting execution of WR_1 using $R1_0$ is suggested. The user follows the recommendation. At time 18, TS_2 is completed. Furthermore, at time 18, starting execution of AS_2 using $R0_1$ is suggested. The user follows the recommendation. At time 20, AT_3 is completed. Furthermore, at time 20, starting execution of AT_4

using $R1_1$ is suggested. The user follows the recommendation. At time 24, WR_1 is completed two time units later than expected. Even with the occurrence of this unexpected event, P_4 is still valid due to the slack time between WR_1 and GE_2. Moreover, at time 24, AS_2 is completed, hence GE_2 becomes enabled. Furthermore, at time 24, starting execution of AS_3 using $R0_1$ and GE_2 using $R1_0$ is suggested. The user follows the recommendation. The remaining activities are executed as expected by considering P_4.

In this way, the recommendation service supports users of flexible PAISs during process execution to optimize the overall process performance goals, by considering optimized enactment plans which are updated when necessary.

6. Empirical evaluation

In order to evaluate the effectiveness of the proposed constraint-based approach, a controlled experiment has been conducted. Section 6.1 describes the design underlying the experiment, and Section 6.2 shows the experimental results and the data analysis.

6.1. Experimental design

In this section, the design underlying the experiment is detailed. The purpose of the empirical evaluation is to analyze the behavior of our proposal in the generation of optimal enactment plans from extended ConDec specifications (including estimates), in order to test the suitability of our proposal for giving recommendations, in terms of performance and quality of recommendations. In particular, we aim to investigate how far different search algorithms are suitable for solving the considered problems. Specifically, three search algorithms are tested for solving the generated models (cf. Section 4.3), i.e., complete search (CP), iterative bounded greedy (IBG), and hybrid search (GRASP-LNS). Since the underlying strategies of the considered search techniques are completely different, we aim to investigate under which circumstances each one is the most suitable for obtaining the solution of certain problems. Moreover, we would like to find out whether these techniques are complementary, and hence can be used in a combined way for obtaining a good solution.

The suitability of our approach is tested regarding both (1) build-time, i.e., generation of complete optimized plans before starting the BP enactment; and (2) run-time, i.e., generation of (partial) optimized plans by considering the actual partial trace of the process as the execution of the process proceeds. The run-time experiments have the goal of analyzing the behavior of the proposed constraint-based approach in cases that the generation of new optimized enactment plans is required due to user deviations or incorrect estimations which make the plans which have been previously generated useless. In this way, for performing these experiments, the constraint-based search algorithms (cf. Section 4.3) do not use any information about plans which have been previously generated. Specifically, for generating plans considering that a specific partial trace has been performed: (1) values to certain constraint-based variables of the proposed CSP are given (i.e., those related to the performed partial trace), and (2) the proposed search algorithms are used to state values for the remaining variables which are unknown (i.e., those related to the remaining trace). As explained in Section 3.2, the proposed recommendation system could consider the plans which have been previously generated when doing replanning in order to improve the efficiency. However, for the run-time experiments we did not use information about previous plans in order to check the behavior of the approach in the worst case, i.e., when a good plan needs to be swiftly generated and any plan which has been previously generated cannot be used since, for example, the user completely disregards all recommendations.

Objects: The empirical evaluation considers different ConDec models taking some important characteristics into account in the generation of the models: (i) correctness, i.e., the ConDec models must represent feasible problems without conflicts (i.e., there are some traces that satisfy the model) and without any dead activities (i.e., none of the traces that satisfies the model contains this activity), (ii) representativeness, i.e., the ConDec models must represent problems which are similar to actual BPs. Consequently, we require the test models to be of medium-size (i.e., including 10–20 activities) and comprise all three types of ConDec templates, i.e., existence, relation, and negation (cf. Section 2.1). Overall, 6 generic test models are considered with 10 and 20 activities respectively and a varying number of constraints (cf. Table 2). Fig. 7 shows the ConDec representation of the generic models 10*A*, 10*B*, 10*C*, 20*A*, 20*B*, and 20*C*.

During the experiments these generic ConDec models are specified by instantiating the generic relationships with concrete constraints leading to different ConDec problems. Since the filtering rules (cf. [3]) for the different ConDec constraints significantly vary in their computational complexity (cf. Table 3 for the different complexity groups), we ensured that the specific ConDec models which are generated cover all complexity groups, i.e., all the possible combinations of type and complexity. For this, several types of problems are considered by including templates of the following groups (cf. Table 3): {E1, R2, N4}, {E1,

Table 2
Generic constraint-based models with 10 and 20 activities and a varying number of constraints used for the empirical evaluation.

Model	#Acts	Description
M10A	10	Includes 10 activities and 4 constraints
M10B	10	Extends M10A by including 3 additional constraints
M10C	10	Extends M10B by including 4 additional constraints
M20A	20	Includes 20 activities and 9 relationships
M20B	20	Extends M20A by including 6 additional constraints, similar to M10B
M20C	20	Extends M20B by including 8 additional constraints, similar to M10C

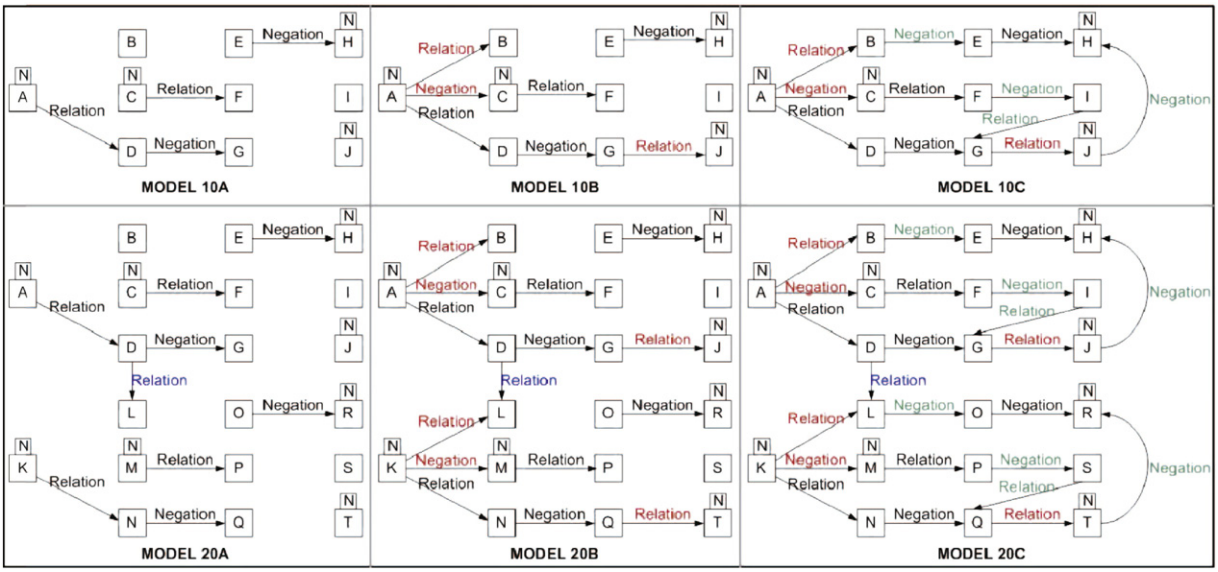


Fig. 7. Generic ConDec models.

R2, N5}, {E1, R2, N6}, {E1, R3, N4}, {E1, R3, N5}, and {E1, R3, N6}. Specifically, for each group of templates, a representative item (in bold in Table 3) is selected for the tests. Moreover, in the case of *Existence* templates, a value for label N must be established ($N \in \{10,20,30,40,50\}$ is considered). Regarding the number of available resources, in turn, for all the generated test models, two available resources of two kinds of roles (i.e., $R1$ and $R2$) are considered. In addition, different durations and required resources for each BP activity are considered (game, G), since these aspects have a great influence on the complexity of the search of optimal solutions due to the fact that the considered problems are an extension of typical scheduling problems. Specifically, in order to average the results over a collection of randomly generated ConDec models, 30 instances are randomly generated for each specific ConDec model by varying activity durations between 1 and 10 and role of required resources between $R1$ and $R2$.

Independent variables: Considering the generated test models and search algorithms (cf. Section 4.3), Table 4 depicts the independent variables which are considered for the empirical evaluation.

Response variables: As stated, the suitability of our approach is tested regarding both build-time and run-time phases, analyzing different response variables.

1. Build-time phase. For the build-time evaluation, a 5-minute time limit is established, since, in general, the initial optimized plans are generated prior to BP enactment, i.e., the search algorithms can be run for a long time. The optimality of the initially

Table 3
Type and complexity of the filtering rules related to the ConDec templates.

Template	Type	Complexity	Group
ExistenceN(A)	**Existence**	$\Theta(1)$	**E1**
AbsenceN(A)	Existence	$\Theta(1)$	E1
ExactlyN(A)	Existence	$\Theta(1)$	E1
Responded Existence(A,B)	Relation	$O(n)$	R2
CoExistence(A,B)	Relation	$O(n)$	R2
Precedence(A,B)	Relation	$O(n)$	R2
Response(A,B)	**Relation**	$\boldsymbol{O(n)}$	**R2**
Succession(A,B)	Relation	$O(n)$	R2
Alternate Precedence(A,B)	Relation	$O(n \times nt^3)$	R3
Alternate Response(A,B)	**Relation**	$O(n \times nt^3)$	**R3**
Alternate Succession(A,B)	Relation	$O(n \times nt^3)$	R3
Chain Precedence(A,B)	Relation	$O(n \times nt^3)$	R3
Chain Response(A,B)	Relation	$O(n \times nt^3)$	R3
Chain Succession(A,B)	Relation	$O(n \times nt^3)$	R3
Responded Absence(A,B)	Negation	$O(n)$	N4
Negation Response(A,B)	**Negation**	$\boldsymbol{O(n)}$	**N4**
Negation Alternate Precedence(A,B)	Negation	$O(n \times nt^2)$	N5
Negation Alternate Response(A,B)	**Negation**	$O(n \times nt^2)$	**N5**
Negation Alternate Succession(A,B)	Negation	$O(n \times nt^2)$	N5
Negation Chain Succession(A,B)	**Negation**	$O(n \times nt^3)$	**N6**

Table 4
Independent variables which are considered for the empirical evaluation.

Name	Description	Values
Model	Generic ConDec model	{*M10A*, *M10B*, *M10C*, *M20A*, *M20B*, *M20C*}
Relation	Value for the label *Relation* in the specific ConDec model	{*Response*, *AlternateResponse*}
Negation	Value for the label *Negation* in the specific ConDec model	{*NegationResponse*, *NegationAlternateResponse*, *NegationChainSuccession*}
N	Value for the label *N* in the specific ConDec model	{10,20,30,40,50}
G	Game containing duration and required resource for each BP activity	{1,2,…,30}
Search	Search algorithms used for solving the generated models	{*CP*,*IBG*,*GRASP-LNS*}

generated plans is considered as a relevant aspect for giving good recommendations. Furthermore, the time spent for obtaining the optimal plans is a rather relevant response variable to be analyzed. The performance measures depicted in Table 5 are studied.

2. Run-time phase. Assuming that the actual trace deviated (e.g., user deviated) from the initially generated optimized execution plan, replanning has to be conducted and new optimized execution plans have to be obtained for the partial trace executed so far. The focus of the run-time evaluation is therefore to test the quality of solutions (i.e., the quality of BP enactment plans, cf. Definition 11) that can be found for given partial traces when replanning is needed due to deviations.

Since at run-time the user is probably expecting for a recommendation the search algorithms for generating optimized plans swiftly need to find a suitable solution. Therefore, for the run-time evaluation, a short time limit (i.e., 5-s) is established. In order to mimic that the deviations happen at different stages in the process, i.e., early versus late, partial traces of different sizes (0%, 25%, 50%, and 75% of the overall process enactment) are considered.

For generating the partial traces for the experiments, several strategies can be used. We generated the partial traces by considering the initial parts of the best plans which have been previously generated by any search algorithm (cf. Section 4.3). The reasons for taking initial parts of the best plans include (1) to easily get feasible traces and (2) to be able to compare the quality of the plans which are generated in 5-s versus the ones which are generated in 5-min. The selection of the traces clearly has an impact on the overall completion time (i.e., *OCT*). However, we believe that this does not bias our results, since we are not interested in absolute overall completion times, but we rather wanted to show that the constraint-based approach is able to find (optimized) execution plans for given partial traces even within 5 s when any plan which has been previously generated is useless. As explained before (cf. second paragraph of this section), we simulate that any plan which has been previously generated is useless by managing the partial traces in the same way as if they were completely new. Therefore, for performing these experiments the constraint-based search algorithms (cf. Section 4.3) do not use any information about plans which have been previously generated with the goal of checking the behavior of the approach in the worst case, i.e., when a good plan needs to be swiftly generated and any plan which has been previously generated is useless.

The performance measures depicted in Table 6 are studied.

For both build-time and run-time evaluation, the results which are obtained by each search technique are compared in order to obtain information about which technique is better in general, or to solve some specific kinds of problems.

Experimental design: For each of the 6 generic models, 150 problem instances are generated considering different values for variable N (5 values) and G (30 values which represent the 30 games which are randomly generated by varying activity durations and role of required resources) using each of the search algorithms. The response variables are then calculated by considering average values for all 150 problem instances.[8]

Experimental execution: The machine for all experiments is an Intel Core2, 2.13 GHz, 1.97 GB memory, running on Windows XP. In order to solve the constraint-based problems (cf. Section 4), the system COMET [10] is used, which is able to generate high-quality solutions for highly constrained problems in an efficient way.

6.2. Experimental results and data analysis

6.2.1. Build-time

For the experiments of the build-time phase, the constraint-based search algorithm is run until a 5-minute CPU time limit is reached.

Fig. 8 shows for each search algorithm: (a) the average percentage of optimal solutions which are found, (b) the average time for getting optimal solutions, considering the cases in which the optimal solution is found, and (c) the average percentage of cases in which a specific technique gets a better solution (i.e., less overall completion time) than the other ones (response variables *%Opt*, T_{Opt}, and *%Best* respectively) versus the problem to be solved. For all the tables and figures presented in this section the considered problems are grouped according to the generic model, i.e., M10A, M10B, M10C, M20A, M20B, M20C, and the response

[8] However, notice that problems stated by a simple model with a high value for *N* can entail a higher complexity than complex models with lower value for *N*.

Table 5
Response variables which are considered for the empirical evaluation at build-time.

Name	Description
$\%Opt$	Average percentage of optimal solutions which are found by each technique (cf. Fig. 8(a))
$\%Opt_{ANY}$	Average percentage of optimal solutions which are found by any technique (Table 8)
T_{Opt}	Average time for getting optimal solutions for each technique, considering the cases in which the optimal solution is found (cf. Fig. 8(b))
$\%Best$	Average percentage of cases in which a specific technique gets a better solution (i.e., less overall completion time) than the other ones (cf. Fig. 8(c))

variable is represented versus specific problems which are obtained after instantiating the relations and negations of the generic model as stated in Table 7, i.e., 6 specific problems ({1, …,6}) for each generic model are considered.

Fig. 8(a) shows that CP search reaches the optimum solution for a greater number of problems than IBG and GRASP-LNS when the complexity of the problems is low, i.e., for models M10A, M10B and M10C. As the complexity of the problem increases, IBG and GRASP-LNS reach the optimal solutions for a greater number of problems than CP, i.e., M20A, M20B and M20C, presenting the highest difference for model M20A. These results can be explained by the fact that CP is a good strategy for getting the optimal solutions for problems which present a low complexity, since for small problems almost the entire search space can be explored. However, for problems with high complexity, an exhaustive search is not a good strategy for searching the optimal solution, since the search area explored by CP is not diversified enough and hence only a part of the possible solutions is analyzed in a limited time. Thus, incomplete techniques are better in general. Furthermore, the evaluation shows that in most cases, IBG outperforms GRASP-LNS. This result can be explained by the fact that the considered IBG is implemented in an efficient way due to the considered heuristic, and GRASP techniques require greater parameter tuning [13,14], and hence more tests need to be done to get the best setting for this technique.

On the other hand, regarding the results which can be obtained by combining all the search algorithms, Table 8 shows the average number of optimal solutions which are found by any of the techniques (response variable $\%Opt_{ANY}$) versus the problem to be solved. It can be seen that for models M10A and M10C, the percentage of optimal solutions which are found is rather high (more than 77% in all cases), regardless of the relationships which are given between the activities. However, for models M20A, M20B, and M20C, this measure highly depends on the relationships which are given between the BP activities, finding the fewest values when alternate response (relationships 4–6) is given. On the other hand, for relationships 1 and 3, the percentage of optimal solutions which are found is rather high (more than 78% in all cases), regardless of the specific model. However, for relationships 4, 5 and 6, this measure highly depends on the model, finding the fewest values for models of 20 activities. Taking these results into account, in general, the specific *Relation* and the number of activities of the BP model seem to be much more influential than the other considered aspects. Furthermore, the average value for all cases is 72.94%, the minimum value is 23.33%, and the maximum value is 100%, which can be considered rather good results.

Fig. 8(b) shows that for all search algorithms, for the same model the response variable greatly increases as the complexity of the filtering rules increases. Furthermore, for the same filtering rules the response variable greatly increases as the complexity of the models increases. Moreover, in most cases, CP reaches the optimum swifter than the other techniques. In turn, GRASP, seems to be the slowest of the three techniques. For CP search, most of these optimums are obtained for the simplest models (cf. Fig. 8(a)), and hence, less time is required. In general, it can be seen that the average time for getting optimums is less than 100 s, which can be considered a rather good result.

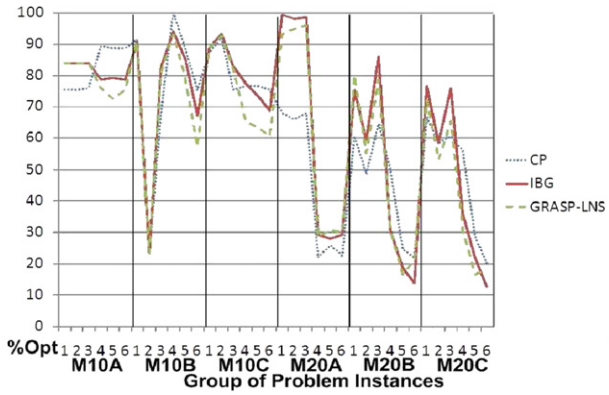
Fig. 8(c) shows that for all search algorithms, for the same model the response variable increases as the complexity of the filtering rules increases, i.e., all techniques present a similar behavior regarding getting optimums for low-complex filtering rules and this behavior highly differs as the complexity of the filtering rules increases. Furthermore, for the same filtering rules the response variable greatly increases as the complexity of the models increases, i.e., all techniques present a similar behavior regarding getting optimums for low-complex models and this behavior highly differs as the complexity of the models increases. Therefore, the use of the considered techniques in a coordinated way is highly recommendable for complex problems, since their behavior is highly different and their result can be combined for obtaining solutions of high quality.

In brief, various conclusions can be drawn after analyzing the results:

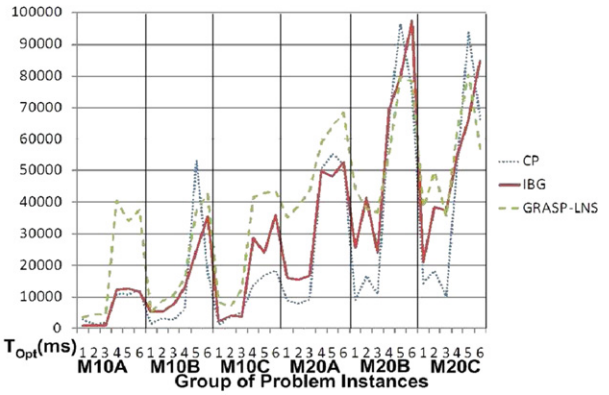
- CP search obtains better results than IBG and GRASP-LNS for simple problems (cf. Fig. 8(a)).
- IBG and GRASP-LNS searches obtain better results than CP for complex problems (cf. Fig. 8(a)).

Table 6
Response variables which are considered for the empirical evaluation at run-time.

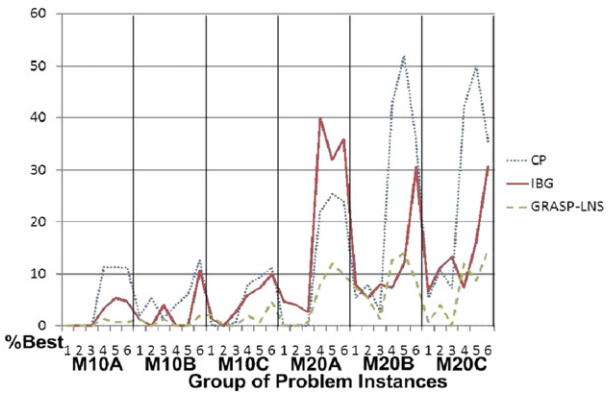
Name	Description
$\%Q_X, X \in \{0,25,50,75\}$	Average quality of solutions which are found by each technique regarding the best solution which is known for each problem after X% of the optimized plan is executed (cf. Fig. 9)
$\%Q_{ANY}$	Average quality of solutions which are found by any technique regarding the best solution which is known for each problem after 0%, 25%, 50% and 75% of the optimized plan is executed (cf. Fig. 10)



(a) Average percentage of optimal solutions found (5-minutes time limit)



(b) Average time for getting optimal solutions



(c) Average percentage of cases a specific technique gets a better solution than the other ones

Fig. 8. Experimental results regarding build-time.

- The percentage of optimum solutions is very high for almost all cases when considering all techniques (Table 8).
- The average time for getting optimums is quite low (cf. Fig. 8(b)).
- The use of a combination of all techniques in a coordinated way highly increases the quality of the solutions, especially for complex problems.

6.2.2. Run-time

For the experiments of the run-time phase, the constraint-based search algorithm is run until a 5-second CPU time limit is reached.

Figs. 9 and 10 show the average quality of solutions which is found when compared with the best known solution for each problem, after 0%, 25%, 50%, and 75% of the BP optimized enactment plan is executed because users, for example, deviated

Table 7
ID for each instantiation of the generic relationships (i.e., relations and negations) of the generic models (cf. Tables 2, 7) which is considered for the experiments.

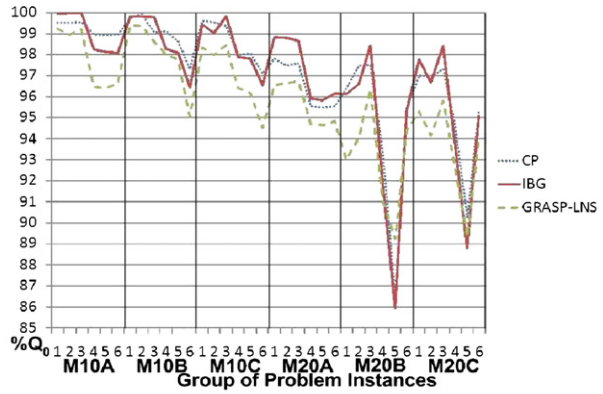
ID	Relation – Negation
1	Response – Negation response
2	Response – Negation alternate response
3	Response – Negation chain succession
4	Alternate Response – Negation response
5	Alternate Response – Negation alternate response
6	Alternate Response – Negation chain succession

Table 8
Average percentage of optimal solutions found in 5-minute time limit when considering all techniques.

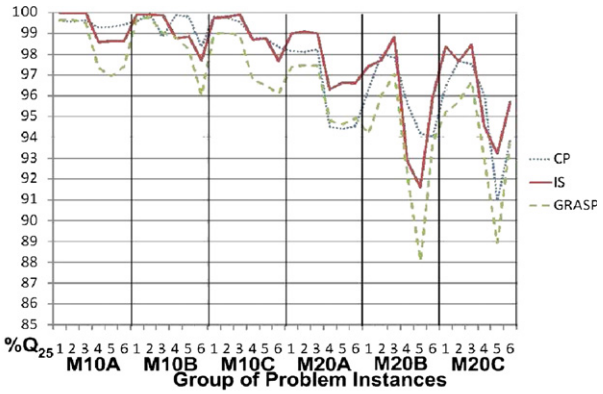
Relation – Negation	M10A	M10B	M10C	M20A	M20B	M20C
1 Resp. – Neg. resp.	84	95.33	88.66	99.33	87.33	78.66
2 Resp. – Neg. alt. resp.	84	23.33	93.33	98	62	67.33
3 Resp. – Neg. chain succ.	84	86	83.33	99.33	93.33	87.33
4 Alt. Resp. – Neg. resp.	89.33	100	80	41.33	50.66	58.66
5 Alt. Resp. – Neg. alt. resp.	89.33	92.66	77.33	43.33	30	32
6 Alt. Resp. – Neg. chain succ.	88.66	76.66	80.66	44	30	26.66

(response variables $\%Q_0$, $\%Q_{25}$, $\%Q_{50}$, $\%Q_{75}$, $\%Q_{ANY}$) versus the problem to be solved. When comparing the different search techniques, similar results compared to the build-time evaluation are obtained due to the previously explained reasons. Moreover, for all search algorithms, it is possible to see that for the same model the quality decreases as the complexity of the filtering rules increases, obtaining the lowest values for relationships 5 and 6 in most cases. Furthermore, for the same filtering rules the quality decreases as the complexity of the models increases, obtaining the lowest values for models M20B and M20C. It should be emphasized that, as stated, in general replanning is less time consuming than initial planning, since CSP variable values become known as execution proceeds. Therefore, the quality of the solutions increases as BP execution proceeds (cf. Fig. 10).

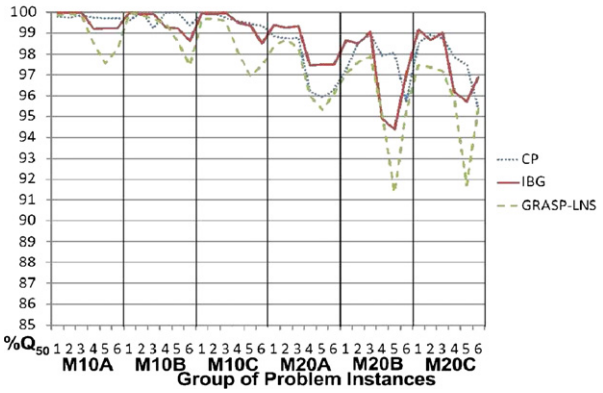
In general, the quality of the solutions which is obtained applying each technique individually is quite good (greater than 86% for all cases). However, the use of all techniques in a coordinated way increases the quality of the solutions even further (greater than 92% for all cases), especially for complex problems (cf. Fig. 10).



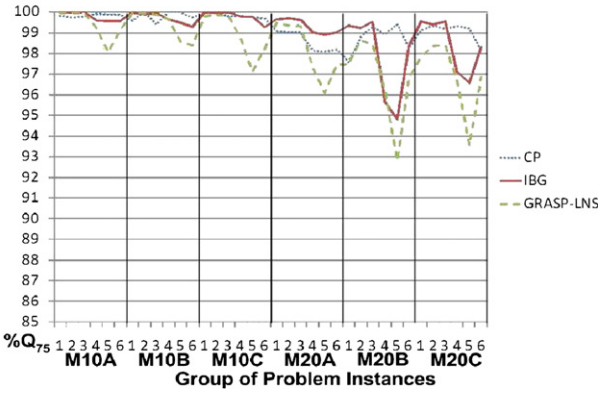
(a) Average quality of solutions which are found after 0% of the BP enactment



(b) Average quality of solutions which are found after 25% of the BP enactment



(c) Average quality of solutions which are found after 50% of the BP enactment



(d) Average quality of solutions which are found after 75% of the BP enactment

Fig. 9. Experimental results regarding run-time: average quality of solutions which are found after 0%, 25%, 50%, and 75% of the BP enactment.

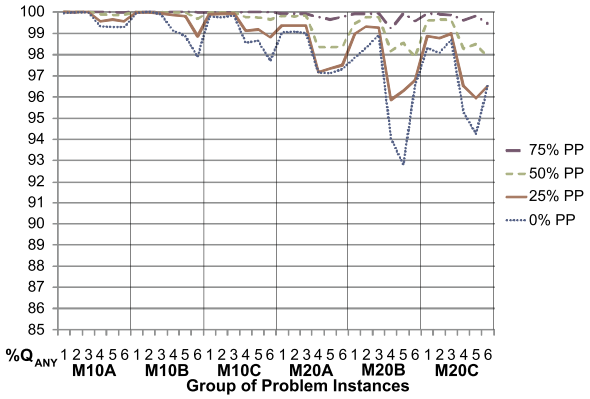


Fig. 10. Average quality of solutions which are found by any technique after 0%, 25%, 50% and 75% of the BP enactment.

7. Discussion

One advantage of our proposal is that the recommendation service is based on optimized enactment plans which are generated by P&S all BP activities, hence it allows for a global optimization of the performance goal. Moreover, the generation of the optimized plans is carried out through a constraint-based approach, which is suitable for modeling and solving P&S problems [40]. In addition, this approach allows modeling the considered problems in an easy way, since the considered declarative specifications are based on high-level constraints. Furthermore, BPs are specified in a declarative way, which is an important step towards the flexible management of PAISs. Moreover, our approach, as an extension of other similar works [41,20], considers the resource perspective besides the control-flow perspective, hence greater optimization can be obtained. Additionally, in order to consider deviations in the estimates, the optimized plans can be updated if necessary, allowing reaction to changes in a quick and flexible way. The generated optimized execution plans can be used for further purposes besides giving recommendations to users of flexible PAISs, e.g., model verification, simulation or time prediction in BP management (i.e., BPM) systems. Moreover, a similar approach can be proposed for imperative models. In that case, the focus would be on recommendations related to resource allocation only (i.e., which activity is using which resource and when, hence influencing the ordering of activity execution). Therefore, the complexity of the constraint-based approach would considerably decrease since the activities to be executed do not need to be stated, i.e., the planning part of the problem would be removed, resulting in a scheduling problem. Note that the complexity of this problem is still very high since the related scheduling problem is NP complete [17].

On the other hand, the proposed approach presents some drawbacks. First, the business analysts must deal with a non-standard language for the declarative specification of BPs, therefore a period of training is required to let the business analysts become familiar with ConDec specifications. Secondly, the optimized plans are generated by considering estimated values for activity durations and resource availabilities, hence our proposal is only appropriate for processes for which the duration of the activities and resource availabilities can be estimated. However, as stated earlier, in the enactment phase, the replanning module can update the current plan by considering the current values of the estimates. Moreover, the considered constraint-based specifications deal with both control-flow and resource perspectives, but do not consider the data perspective. It is intended to consider this aspect in our future work. Furthermore, in this work, only the basic ConDec templates introduced in [46] are considered. ConDec templates can be easily extended so that the specification of more complex and flexible problems can be stated. As future work, some extensions are intended to be considered in the proposed constraint-based approach, e.g., branched templates (i.e., high-level relationships given between more than two activities). Moreover, additional evaluations of our proposal in the context of real process executions are planned as future work.

In our proposal we focus on the minimization of overall completion time. However, it can be easily extended in order to consider further objectives, such as cost or other temporal measures.

8. Related work

Several research groups have integrated AI P&S techniques with BPM systems. Most of the related work integrates scheduling tools into the BPM systems in the enactment phase, in order to make dispatching decisions as to which activity should be executed using a resource when it becomes free (dynamic scheduling) [56,42,19,34,44]. Moreover, [21,4] support resource allocation in business process execution when considering performance optimization. Other authors use AI for exception handling during run-time [37,15]; we, in turn, use a constraint-based modeling approach and make suggestions to users. Furthermore, unlike our work, the related proposals rely on imperative specifications. Notice that integrating AI and BPM is not new. However, the use of AI P&S techniques for giving user recommendations is a new application area.

Related to decision support systems, [28] develops a knowledge-based decision support system (KBDSS) for short-term scheduling in flexible manufacturing systems (FMS). Unlike our proposal, [28] considers the optimization of the efficient use of the machining cells by using a knowledge-based expert system to support the decision making process. Moreover, [7] manages multiple objectives in a hierarchical way. In a related way, [43] addresses the scheduling of a group of employees with different productivity

considering the stochastic nature of customer arrivals. Furthermore, [36,24] propose business process simulation for operational decision support through process mining. While all approaches [28,7,36,24] are based on the knowledge which has been learnt in prior executions, our approach is based on a constraint-based approach for the generation of optimized plans by considering estimated values. Furthermore, [28,7,43,36,24] do not consider declarative process specifications.

Related to user recommendations, [51] generates recommendations to select the next step to optimize some objective functions. While [51] is based on a product data model for generating the recommendations, our approach is based on optimized enactment plans. In addition, [41,20] support users during the execution of declarative process models by selecting among enabled activities. Unlike our work, the recommendations are based on similar past process executions, instead of optimized plans which are generated through a constraint-based approach. Moreover, while our approach allows for a global optimization of the performance goal, the recommendations described in [41,20] bear the risk of creating local optimums. Furthermore, [51,41,20] only consider the control-flow perspective, while our approach also deals with the resource perspective. All of the previously mentioned approaches optimize the execution of single process instances. Our approach, in turn, provides recommendations for optimizing the execution of several instances.

Additionally, there exist some proposals which could be used to generate optimized enactment plans for BPs from declarative process specifications. Specifically, [29] proposes the generation of a non-deterministic finite state automaton from declarative specifications based on linear temporal logic (LTL) which represents exactly all traces that satisfy the LTL formulas. This approach could then be extended, by including estimates, and the overall completion time of all the traces can then in turn be calculated (e.g., [48]). In this approach, the big disadvantage is that the process of generating the automaton from the declarative specifications is NP complete, and, unlike our approach, any heuristic has been used. In a similar way, CLIMB [26] can be used to generate quality traces from declarative specifications, and calculate its completion time. Then, the best traces can be selected. Unlike our approach, [26] does not consider neither optimality nor resource perspective.

9. Conclusion and future work

We propose a recommendation system for giving users assistance during process execution in flexible PAISs to optimize performance goals of the processes (i.e., minimization of overall completion time). The recommendations are generated considering the partial trace of the process execution, the constraint-based process model and resource availabilities. The recommendation system is based on a constraint-based approach, which is used for P&S the activities such that the process goal is optimized. In the proposed approach, both control-flow and resources are considered. Furthermore, the optimized enactment plans are updated by replanning techniques when necessary. Different constraint-based algorithms are applied to a range of test models of varying complexity, including an actual example, so that the suitability of our approach is tested regarding both build-time and run-time phases. The results indicate that the proposed approach produces a satisfactory number of suitable solutions, i.e., solutions which are optimal in most cases and quite good in other cases.

As for future work, it is intended to integrate the proposed recommendation system with a framework for evaluating algorithms for operational support [27]. In addition, the proposed approach is intended to be extended by considering further objective functions. Moreover, we will explore various constraint-based solving techniques and analyze their suitability for the generation of optimized plans to assist users through recommendations. Furthermore, some extensions of ConDec templates are intended to be considered, e.g., branched templates (i.e., high-level relationships given between more than two activities). Additionally, we intend to consider further resource patterns.

Acronyms

AI	artificial intelligence
BP	business process
BPM	business process management
CJSSP	cumulative job shop scheduling problem
COP	constraint optimization problem
CP	constraint programming
CSP	constraint satisfaction problem
GRASP	greedy randomized adaptive search procedure
IBG	iterative bounded greedy
LNS	large neighborhood search
OCT	overall completion time
PAISs	process-aware information systems
P&S	planning and scheduling

Acknowledgements

This work has been partially funded by the Spanish Ministerio de Ciencia e Innovación (TIN2009-13714) and the European Regional Development Fund (ERDF/FEDER).

References

[1] P. Baptiste, C. Le Pape, W. Nuijten, Satisfiability tests and time-bound adjustments for cumulative scheduling problems, Annals of Operations Research 92 (1999) 305–333.
[2] I. Barba, C. Del Valle, A constraint-based approach for planning and scheduling repeated activities, Proc. Coplas, 2011, pp. 55–62.
[3] I. Barba, C. Del Valle, Filtering rules for ConDec templates — pseudocode and complexity. Online http://regula.lsi.us.es/MOPlanner/FilteringRules.pdf 2011, (accessed 08-January-2013).
[4] I. Barba, C. Del Valle, Planning and scheduling of business processes in run-time: a repair planning example, Information Systems Development (2011) 75–87.
[5] I. Barba, B. Weber, C. Del Valle, Supporting the optimized execution of business processes through recommendations, Proc. BPI, vol. 99, 2012, pp. 135–140.
[6] P. Brucker, S. Knust, Complex Scheduling (GOR-Publications), Springer-Verlag New York, Inc., 2006.
[7] A.R. Chaturvedi, G.K. Hutchinson, D.L. Nazareth, Supporting complex real-time decision making through machine learning, Decision Support Systems 10 (2) (1993) 213–233.
[8] P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, A. Zbyslaw, Freeflow: mediating between representation and action in workflow systems, Proc. CSCW, 1996, pp. 190–198.
[9] In: M. Dumas, W.M.P. van der Aalst, A.H. ter Hofstede (Eds.), Process-aware Information Systems: Bridging People and Software Through Process Technology, Wiley-Interscience, 2005.
[10] Dynadec, Comet downloads. Online http://dynadec.com/support/downloads/ 2010, (accessed 08-January-2013).
[11] D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, S. Zugal, Declarative versus imperative process modeling languages: the issue of understandability, Proc. BPMDS 2009 and EMMSAD 2009, 2009, pp. 353–366.
[12] D. Fahland, J. Mendling, H.A. Reijers, B. Weber, M. Weidlich, S. Zugal, Declarative versus imperative process modeling languages: the issue of maintainability, Proc. BPM Workshops, 2010, pp. 477–488.
[13] T.A. Feo, M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.
[14] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.
[15] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, G. Tagni, Exception handling for repair in service-based processes, IEEE Transactions on Software Engineering 36 (2) (2010) 198–215.
[16] H.L. Gantt, Work, Wages, and Profits, Engineering Magazine Co., 1913
[17] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
[18] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, 2004.
[19] B. Ha, J. Bae, Y. Park, S. Kang, Development of process execution rules for workload balancing on agents, Data & Knowledge Engineering 56 (1) (2006) 64–84.
[20] C. Haisjackl, B. Weber, User assistance during process execution — an experimental evaluation of recommendation strategies, Proc. BPI, vol. 66, 2011, pp. 134–145.
[21] Z. Huang, W.M.P. van der Aalst, X. Lu, H. Duan, Reinforcement learning based resource allocation in business process management, Data & Knowledge Engineering 70 (1) (2011) 127–145.
[22] C. Le Pape, P. Couronne, D. Vergamini, V. Gosselin, Time-versus-capacity compromises in project scheduling, Proc. PlanSIG, 1994, pp. 498–502.
[23] C. Li, M. Reichert, A. Wombacher, Mining business process variants: challenges, scenarios, algorithms, Data & Knowledge Engineering 70 (5) (2011) 409–434.
[24] Y. Liu, H. Zhang, C. Li, R.J. Jiao, Workflow simulation for operational decision support using event graph through process mining, Decision Support Systems 52 (3) (2012) 685–697.
[25] R. Lu, S. Sadiq, V. Padmanabhan, G. Governatori, Using a temporal constraint network for business process execution, Proc. ADC, 2006, pp. 157–166.
[26] M. Montali, Specification and verification of declarative open interaction models: a logic-based approach. PhD thesis, Department of Electronics, Computer Science and Telecommunications Engineering, University of Bologna, 2009.
[27] J. Nakatumba, M. Westergaard, W.M.P. van der Aalst, An infrastructure for cost-effective testing of operational support algorithms based on colored Petri nets, Proc. PETRI NETS, 2012, pp. 308–327.
[28] M. Özbayrak, R. Bell, A knowledge-based decision support system for the management of parts and tools in FMS, Decision Support Systems 35 (4) (2003) 487–515.
[29] M. Pesic, Constraint-based workflow management systems: shifting control to users. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2008.
[30] M. Pesic, M.H. Schonenberg, N. Sidorova, W.M.P. van der Aalst, Constraint-based workflow models: change made easy, OTM Conferences (1), 2007, pp. 77–94.
[31] D. Pisinger, S. Ropke, Large neighborhood search, International Series in Operations Research & Management Science 146 (13) (2010) 399–420.
[32] M. Reichert, J. Kolb, R. Bobrik, T. Bauer, Enabling personalized visualization of large business processes through parameterizable views, Proc. SAC, 2012, pp. 1653–1660.
[33] M. Reichert, B. Weber, Enabling Flexibility in Process-aware Information Systems, Springer, 2012.
[34] S. Rhee, N.W. Cho, H. Bae, Increasing the efficiency of business processes using a theory of constraints, Information Systems Frontiers 12 (4) (2010) 443–455.
[35] F. Rossi, P. van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier, 2006.
[36] A. Rozinat, M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, C.J. Fidge, Workflow simulation for operational decision support, Data & Knowledge Engineering 68 (9) (2009) 834–850.
[37] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, Workflow exception patterns, Proc. Caise, 2006, pp. 288–302.
[38] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond, Workflow resource patterns: identification, representation and tool support, Proc. Caise, 2005, pp. 216–232.
[39] I. Rychkova, G. Regev, A. Wegmann, High-level design and analysis of business processes: the advantages of declarative specifications, Proc. RCIS, 2008, pp. 99–110.
[40] M.A. Salido, Introduction to planning, scheduling and constraint satisfaction, Journal of Intelligent Manufacturing 21 (1) (2010) 1–4.
[41] H. Schonenberg, B. Weber, B.F. van Dongen, W.M.P. van der Aalst, Supporting flexible processes through recommendations based on history, Proc. BPM, 2008, pp. 51–66.
[42] J.H. Son, M.H. Kim, Improving the performance of time-constrained workflow processing, Journal of Systems and Software 58 (3) (2001) 211–219.
[43] G.M. Thompson, J.C. Goodale, Variable employee productivity in workforce scheduling, European Journal of Operational Research 170 (2) (2006) 376–390.
[44] C. Tsai, K. Huang, F. Wang, C. Chen, A distributed server architecture supporting dynamic resource provisioning for BPM-oriented workflow management systems, Journal of Systems and Software 83 (8) (2010) 1538–1552.
[45] W.M.P. van der Aalst, S. Jablonski, Dealing with workflow change: identification of issues and solutions, International Journal of Computer Systems Science and Engineering 15 (5) (2000) 267–276.
[46] W.M.P. van der Aalst, M. Pesic, DecSerFlow: towards a truly declarative service flow language, LNCS 4184, 2006, pp. 1–23.
[47] W.M.P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: balancing between flexibility and support, Computer Science — Research and Development 23 (2) (2009) 99–113.
[48] W.M.P. van der Aalst, M.H. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2) (2011) 450–475.
[49] B.F. van Dongen, W.M.P. van der Aalst, A meta model for process mining data, Proc. Caise, 2005, pp. 309–320.
[50] P. van Hentenryck, The OPL Optimization Programming Language, MIT Press, 1999.
[51] I. Vanderfeesten, H.A. Reijers, W.M.P. van der Aalst, Product based workflow support: a recommendation service for dynamic workflow execution, Technical Report BPM-08-03, BPMcenter.org, 2008.
[52] J. Wainer, F. Bezerra, P. Barthelmess, Tucupi: a flexible workflow system based on overridable constraints, Proc. SAC, 2004, pp. 498–502.
[53] J. Wainer, F. De Lima Bezerra, Constraint-based flexible workflows, LNCS 2806, 2003, pp. 151–158.

[54] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change support features — enhancing flexibility in process-aware information systems, Data & Knowledge Engineering 66 (3) (2008) 438–466.
[55] M. Weske, Business Process Management: Concepts, Methods, Technology, Springer, 2007.
[56] J.L. Zhao, E.A. Stohr, Temporal workflow management in a claim handling system, SIGSOFT: Software Engineering Notes 24 (2) (1999) 187–195.