A Benchmark for ASP Systems: Resource Allocation in Business Processes

Giray Havur
Cristina Cabanillas
Axel Polleres

Arbeitspapiere zum Tätigkeitsfeld
Informationsverarbeitung, Informationswirtschaft und Prozessmanagement
Working Papers on Information Systems, Information Business and Operations

Nr./No. 01/2019
ISSN: 2518-6809
URL: http://epub.wu.ac.at/view/p_series/S1/

Herausgeber / Editor:
Department für Informationsverarbeitung und Prozessmanagement
Wirtschaftsuniversität Wien · Welthandelsplatz 1 · 1020 Wien
Department of Information Systems and Operations · Vienna University of Economics and Business · Welthandelsplatz 1 · 1020 Vienna

*A Benchmark for ASP Systems: Resource Allocation in Business Processes**

GIRAY HAVUR[1,2], CRISTINA CABANILLAS[1], AXEL POLLERES[1]
[1]Vienna University of Economics and Business, Vienna, Austria
[2]Siemens AG Österreich, Corporate Technology, Vienna, Austria
(*e-mail:* {name}.{surname}@wu.ac.at)

Abstract

The goal of this paper is to benchmark Answer Set Programming (ASP) systems to test their performance when dealing with a complex optimization problem. In particular, the problem tackled is resource allocation in the area of Business Process Management (BPM). Like many other scheduling problems, the allocation of resources and starting times to business process activities is a challenging optimization problem for ASP solvers. Our problem encoding is ASP Core-2 standard compliant and it is realized in a declarative and compact fashion. We develop an instance generator that produces problem instances of different size and hardness with respect to adjustable parameters. By using the baseline encoding and the instance generator, we provide a comparison between the two award-winning ASP solvers CLASP and WASP and report the grounding performance of GRINGO and I-DLV. The benchmark suggests that there is room for improvement concerning both the grounders and the solvers. Fostered by the relevance of the problem addressed, of which several variants have been described in different domains, we believe this is a solid application-oriented benchmark for the ASP community.

KEYWORDS: answer set programming, benchmark, business process management, resource allocation

1 Introduction

Business process management (BPM) is a discipline in operations management that aims at improving corporate performance by properly managing and optimizing a company's business processes. A business process is a collection of related events, activities and decisions that involve a number of human and non-human resources and that collectively lead to an outcome that is of value to an organization or its customers (Dumas et al., 2018). Resources, as an integral part of business processes, have to be considered throughout all the stages of the BPM lifecycle, which iterates from process discovery and modeling to process execution and monitoring targeting continuous process improvement. At run time, a process execution engine creates instances of a business process and allocates specific resources to the tasks to be completed according to pre-defined criteria. Such criteria are defined in the form of constraints that comprise the characteristics as well as the number of resources needed for completing each process task. When referred to human resources, the characteristics are usually reflected in organizational models that contain all the relevant data about the resources, their roles, their skills and any other valuable

* This work has been funded by the Austrian Research Promotion Agency (FFG) under the project grant 845638 (SHAPE) and the Austrian Science Fund (FWF) under the project grant V 569-N31 (PRAIS).

information. The Role-Based Access Control (RBAC) model is very often used for assigning resources to process tasks on the basis of the organizational roles they are associated with. As a result, during process execution, at the due time for each task only the required number of resources will be picked up from the set of candidates (i.e. suitable resources according to their roles) and allocated to the task. Therefore, we can define resource allocation in business processes as the assignment of resources and time intervals to the activities for the execution of a process. The complexity of resource allocation arises from coordinating the dependencies across a broad set of resources and process activities as well as from solving potential conflicts on the use of certain resources (e.g. certain roles are required to execute certain activities, particular resources cannot be used at the same time, or different resources have to be allocated to different activities to avoid conflicts of interest, i.e. separation of duties) (Russell et al., 2005).

Process-oriented organizations are concerned with carrying out a proper resource allocation as they aim to optimize one or more process performance measures, which include time, cost, quality and flexibility (Dumas et al., 2018). Typically, among these objectives a subset of them is selected and each objective in this subset is assigned a relative priority that defines the respective optimization function. Such a problem is similar to a well-known scheduling problem called the Multi-mode Resource-constrained Project Scheduling Problem (MRCPSP) (Sprecher et al., 1994).

One way of addressing this problem is describing it in a knowledge representation formalism. Answer Set Programming (ASP) appears as a good candidate for this purpose as it has been used to solve various hard computational problems and proved to maintain a balance between expressiveness, ease of use and computational effectiveness (Gebser et al., 2012). Moreover, any possible trade-off related to an allocated resource (e.g. duration of an activity given a resource) can be neatly represented using the defaults in the absence of complete information. As a matter of fact, we have previously used ASP as a flexible tool to encode the resource allocation problem and related complex constraints in prior research efforts (Havur et al., 2015, 2016), which allowes us to identify the problem-in-hand as a challenging task for the ASP systems, and therefore in this paper we describe this problem as a benchmark for the ASP systems.

Earlier ASP competitions have led to the development of two scheduling-related benchmark entries. The first entry is called disjunctive scheduling (Denecker et al., 2009) and the problem addressed has generalized precedence relations, uniform activity durations, an overall deadline to be met with no resource aspect. The other entry is called incremental scheduling (Calimeri et al., 2014) and addresses a problem with one renewable resource set, uniform activity durations and deadlines imposed on activities. Unlike the problems involved in these two entries, the resource allocation problem has different numbers of resource sets, resource- and resource-set-specific time requirements, different resource demands per activity and a makespan[1] optimization function. For such a type of problems, a benchmark to compare the performance of existing ASP solvers and grounders is still missing. In the case of resource allocation in the area of BPM, this is also due to the lack of available data describing a variety of scenarios in which resource allocation is required.

In this paper, we narrow this gap by extending former contributions on ASP-based resource allocation in business processes towards a common challenge benchmark for ASP solvers. In particular, our contribution is three-fold:

[1] The makespan is the distance in time that elapses from the start to the end of a process.

- We position the problem of resource allocation in the landscape of existing scheduling problems by characterizing their activity and resource environments.
- Based on a *declarative baseline encoding in ASP* of the resource allocation problem with RBAC constraints[2] and a standardized encoding of problem instances, we develop an *instance generator* that produces both process instances and organizational resource models along with different parameters that can be used to adapt instance sizes. This sets up the *benchmark* as it provides all the inputs that are needed to test the performance of any ASP grounder/solver when dealing with the resource allocation problem in the BPM domain.
- Lastly, by using our benchmark we present a detailed *evaluation* comparing two of the best performing ASP solvers (CLASP (Gebser et al., 2015) and WASP (Alviano et al., 2017)) and grounders (GRINGO (Gebser et al., 2011) and I-DLV (Calimeri et al., 2017)).

Our benchmark implementation is configurable in the sense that any grounder/solver combinations can be added in. The results of our benchmark evaluation demonstrate that real-world instance sizes pose a considerable challenge on state-of-the-art ASP systems, which opens opportunities for further performance improvements in these systems.

The paper is structured as follows: Section 2 provides necessary background by formally defining the problem of resource allocation in business processes and positioning it within the spectrum of scheduling problems already targeted by ASP experts. Section 3 describes the problem encoding in ASP and the problem instance generator. Section 4 presents the performance evaluation of different ASP systems using our problem encoding and the problem instances produced by our generator. Section 5 reflects on the related work. Section 6 concludes the paper with the gained insights and gives pointers for the future work.

2 Background

The resource allocation problem deals with the assignment of resources and time intervals to the execution of activities, where the number of activities as well as the number of resources are assumed to be finite (Lombardi and Milano, 2012). The complexity of resource allocation in BPM arises from coordinating the explicit and implicit dependencies across a broad set of resources and activities of processes as well as from solving potential conflicts on the use of certain resources while minimizing the *makespan* (Havur et al., 2016). Before introducing our ASP encoding and benchmark setting, in this section we describe the elements involved in this problem and position the problem with respect to other (scheduling) problems.

2.1 Business Processes

A business process is a finite set of activities whose executions are partially ordered. The execution of each activity in the process generally requires one or more (human or non-human) resources and produces some output that is of benefit for a customer, such as a service or a document (Burattin, 2015). Process models are defined to represent the different aspects of a business process, especially the functional (process activities) and the behavioral (control flow or execution order) perspectives. For instance, Figure 1 depicts the model of a process for publishing a book from the point of view of a publishing house, represented using Business Process

[2] To keep the encoding compact, in this paper we allocate only human resources but in prior research we developed a generalized ASP encoding covering different resource types (cf. Havur et al. (2016)).

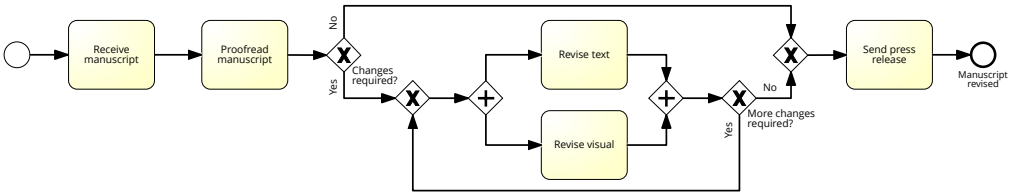


Fig. 1: BPMN model of the book publishing process

Model and Notation (BPMN) (OMG, 2011). In this process, when the publishing entity receives a new textbook manuscript from an author, it must be proofread. If changes are required, the modifications suggested must be applied on text and figures, which can be done in parallel. This review-and-improvement procedure is repeated until there are no more changes to apply, and the improved manuscript is then sent back to the author for double-checking. Although BPMN is the de-facto standard process modeling notation due to its understandability and ease of use, in this work we rely on time Petri nets (Popova-Zeugmann, 2013) satisfying the workflow properties (van der Aalst, 1996) for process modeling because of their well-defined semantics and their analysis capabilities. Nonetheless, note that many process modeling notations, including BPMN, can be automatically mapped to Petri nets (e.g. by implementing the transformations defined by, e.g. Lohmann et al. (2009)). A Petri net is a directed graph with two kinds of nodes, interpreted as places and transitions, such that no arc connects two nodes of the same kind. The places of the Petri net might contain tokens, whose distribution might change over time, giving rise to the net behavior.

Definition 1 (*Petri net*)
A *Petri net* is a 5-tuple $PN = (P,T,F,M_0,\nu)$, where:

- $P = \{p_1, p_2, ..., p_n\}$ is the set of *places*, represented graphically as circles,
- $T = \{t_1, t_2, ..., t_n\}$ is the set of *transitions*, represented graphically as rectangles,
- $F \subseteq (P \times T) \bigcup (T \times P)$ is the set of *arcs* (flow relations), represented as arrows and describing a bipartite graph,
- M_0 is the *initial marking*, where a *marking* (or *state*) $M = \{\mu(p_1), \mu(p_2), ..., \mu(p_{|P|})\}$, $\mu : P \to \mathbb{Z}_{\geq 0}$, represents the distribution of tokens over the set of places. When it assigns a non-negative integer k to place p, we say that p is marked with k tokens. Pictorially, we place k black dots (tokens) in place p; and
- $\nu : F \to \mathbb{Z}^+$ is the arc weight mapping indicating cardinality constraints on the movement of tokens throughout the net.

The *input places* and the *output places* of each transition $t \in T$ are ${}^\bullet t = \{p \in P \mid (p,t) \in F\}$ and $t^\bullet = \{p \in P \mid (t,p) \in F\}$, respectively. Similarly, the *input transitions* and the *output transitions* of each place $p \in P$ are ${}^\bullet p = \{t \in T \mid (t,p) \in F\}$ and $p^\bullet = \{t \in T \mid (p,t) \in F\}$, respectively. The initial marking of the Petri net M_0, which represents the initial distribution of tokens over the places, can change into successor markings. These changes are described as *firing rules*. Such rules introduce a dynamic aspect to the Petri net by allowing the firings to modify the state of the Petri net. The minimum number of necessary tokens on the places for enabling a transition t is defined by $\nu(p,t)$. A transition t is *enabled* when there are at least as many tokens as $\nu(p,t)$ in each input place $p \in {}^\bullet t$. An enabled transition can therefore *fire*. The number of tokens which are added to each output place after firing of t is defined as $\nu(t,p)$. The firing of a transition changes

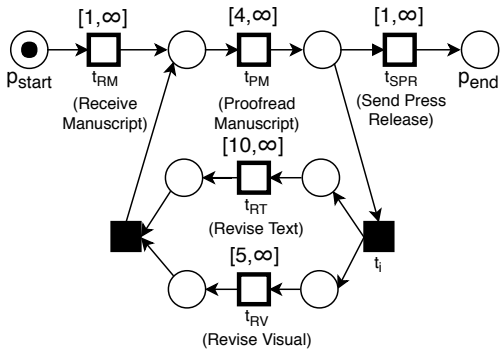


Fig. 2: Petri net model of the book publishing process

the current marking by subtracting $v(t,p)$ amount of tokens from each input place $p \in {}^\bullet t$ and adding $v(p,t)$ amount of tokens to each output place $p \in t^\bullet$, and hence, it moves the net from a marking M_{k-1} to a new marking M_k denoted as $M_{k-1} \xrightarrow{t_k} M_k$. A firing sequence of transitions $\sigma = t_1, t_2, ..., t_n$ changes the state of the Petri net at each firing: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} ... \xrightarrow{t_n} M_n$. A marking M_k is *reachable* if there is a sequence σ such that $M_0 \xrightarrow{\sigma} M_k$ (i.e. from the initial marking to M_k).

Petri nets are classified according to several criteria, including cardinality and behavioral constraints. The Petri nets that represent the business processes addressed in this work have the following properties:

- They constitute a so-called *workflow net* (van der Aalst, 1996), which means that they contain a starting place p_s such that ${}^\bullet p_s = \emptyset$ and an ending place p_e such that $p_e^\bullet = \emptyset$; and they are *connected*, i.e. every node in the Petri net is on the path from p_s to p_e.
- They are *1-safe*, which means that each place contains at most one token at any state (i.e. for any place $p \in P$, $0 \leq \mu(p) \leq 1$).
- They are *free-choice*, which implies that the choice between multiple transitions can never be influenced by the rest of the net (i.e. for any two different places $\{p_i, p_j\} \subseteq P$, $(p_i^\bullet \cap p_j^\bullet) = \emptyset$ or $p_i^\bullet = p_j^\bullet$),
- Their transitions are labelled with a minimum and a maximum waiting time in order to be able to represent the temporal aspects related to activities needed for resource allocation, i.e. *Time Petri nets* (TPN) (Popova-Zeugmann, 2013).

Definition 2 (*Time Petri net*)
A time Petri net is a 6-tuple $TPN = (P, T, F, M_0, v, \Delta)$, where:

- P, T, F, M_0, v constitute a Petri net, and
- $\Delta : T \to \mathbb{Z}_{\geq 0} \times (\mathbb{Z}_{\geq 0} \cup \{\infty\})$ is the interval function of the TPN, where $\Delta(t) = [\delta_{min}(t), \delta_{max}(t)]$. $\delta_{min}(t)$ and $\delta_{max}(t)$ are the minimum and maximum waiting time to fire t in a use-case specific time unit (TU) (e.g. in minutes, hours or days).

In time Petri nets, process activities correspond to timed transitions. The periods that timed transitions (we will call them *activity transitions*) are enabled (i.e. waiting times) represent activity durations and transition firings indicate activity completions. If the maximum waiting time of t is ∞, it means that the firing of t is not enforced to occur at any time.

Figure 2 illustrates the time Petri net representation of the book publishing process described above, extended to reflect the minimum duration estimations of the process activities (i.e. $\delta_{min}(t)$).

The activity transitions are graphically represented by empty squares along with their minimum and maximum waiting times on top. On the contrary, the *immediate transitions* are shown as filled squares and they fire immediately (i.e. their earliest firing time and latest possible firing time are both equal to 0). These are used for representing concurrent activities in process executions (e.g. t_i in Figure 2).

Four types of ordering relations can be identified between transitions of a Petri net. Specifically, given two transitions $t_x, t_y \in T$ of a Petri net, then:

- t_x *directly precedes* t_y, denoted as $t_x \rightarrow t_y$, if the net contains a path with *exactly* two arcs (i.e. $(t_x, p_i) \cup (p_i, t_y) \subseteq F$) leading from t_x to t_y (e.g. t_{PM} and t_i in Figure 2). This precedence relation is reduced to cover activity transitions in the *direct activity precedence* relation, denoted as $t_x \xrightarrow{a} t_y$, such that $t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n$ where only t_0 and t_n are activity transitions (e.g. $t_{PM} \xrightarrow{a} t_{RT}$ and $t_{PM} \xrightarrow{a} t_{RV}$ in Figure 2).
- t_x *precedes* t_y, denoted as $t_x \Rightarrow t_y$, if the net contains a path with *at least* two arcs leading from t_x to t_y (e.g. t_{RM} and t_{SPR} in Figure 2).
- t_x *is in conflict with* t_y, denoted as $t_x \# t_y$, if the net contains two paths leading to t_x and t_y, respectively, which start at the same place and immediately diverge (e.g. t_{SPR} and t_i in Figure 2).
- t_x *can be executed parallelly with* t_y, denoted as $t_x || t_y$, if t_x and t_y are neither in preceding nor in excluding relation (e.g. t_{RT} and t_{RV} in Figure 3).

The conflicting transitions are necessary to describe a choice (or decision) point in process executions. For example, in Figure 2 it is not known before running the process if the manuscript will be ready to be sent for the press release or it will require (other) revision rounds. However, this kind of uncertainty that causally divides the search space prevents us from allocating resources and starting times to activities in a global time-optimal way. In order to avoid this, we first obtain an *instance* of the unfolding (i.e. a partial run of the net) of the process which contains no conflicts by making *assumptions* on these choices ahead of time, as explained next.

Unfolding (Bonet et al., 2008) is a method used for developing processes with no conflicts. It generates possible firing sequences of the given Petri net $PN = (P, T, F, M_0, \nu)$ while it maintains the partial order of transitions based on the precedence relation induced by the net. It produces a pair $\beta = (ON, \varphi)$, where $ON = (P_B, T_E, F')$ is an *occurrence net*, which is a Petri net without conflicts and cycles, and $\varphi : P_B \cup T_E \rightarrow P \cup T$ is a homomorphism from ON to PN.

Definition 3 (*Occurrence net*)
An *Occurrence net* $ON = (P_B, T_E, F')$ is a net where P_B is the set of *conditions* (cf. places), T_E is the set of *events* (cf. transitions) and F' are the arcs between P_B and T_E defined as follows:

- $\forall b \in P_B$, $|{}^\bullet b| \leq 1$ and N is acyclic,
- $\forall e \in {}^\bullet P_B$, the set $b \in {}^\bullet T_E$ is finite, and
- $\nexists e \in T_E, e \# e$ (i.e. there is no event in self-conflict).

The process of unfolding breaks all reachable places in conflict by making independent copies of such places. For this reason, ON may contain multiple copies of the nodes in the original net. In most cases, β is infinite and cannot be built, therefore it requires a *cut-off event* to halt. We refer the reader to (Bonet et al., 2008) for further details. In order to rematch the conditions and events back to places and transitions, we introduce the concept of *branching net* (Couvreur et al., 2013).

Definition 4 (*Branching net*)

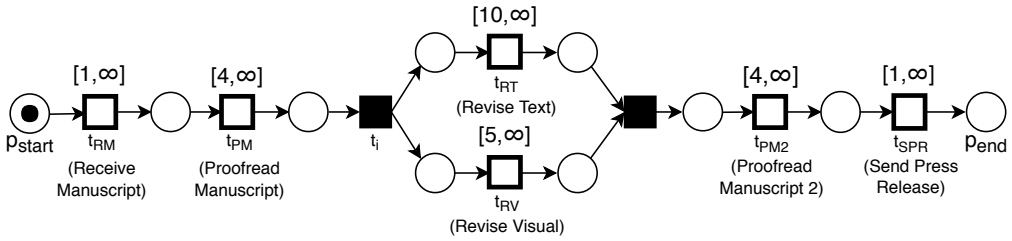


Fig. 3: Unfolded process to publish a book with the assumption that the loop is taken once

A branching net of a Petri net system $PN = (P, T, F, M_0, \nu)$ is a labeled occurrence net $ON_\lambda = (P_B, T_E, F', \lambda)$ where the labeling function λ satisfies the following properties:

- $\lambda(P_B) \subseteq P$ and $\lambda(T_E) \subseteq T$,
- $\forall p \in P$, $\exists b \in P_B$, $\lambda(p) = b$ (i.e. λ is a surjection),
- $|\{b \in P_B : \mu_0(\lambda(b)) = 1, \lambda(b) = p\}| = 1$,
- $\forall e_1, e_2 \in T_E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $\lambda(e_1) = \lambda(e_2)$, then $e_1 = e_2$.

After obtaining a branching net ON_λ from a given net, we lastly introduce *assumption set* to decompose ON_λ and thus generate the *conflict-free* instance of the net PN. An assumption set is described as $\mathscr{A} \subseteq 2^{(T \times \mathbb{Z}_{\geq 0})}$, which constraints the cardinality of events in the selected subnet that starts from $b_s \in P_B$, $\lambda(b_s) = p_s$ and ends at $b_e \in P_B$, $\lambda(b_e) = p_e$, that is, $ON'_\lambda = (P'_B, T'_E, F'', \lambda)$ satisfies (1) $\forall (t_i, n_i) \in \mathscr{A}$:

$$|T'_{E_{t_i}}| = n_i \qquad \text{where } T'_{E_{t_i}} = \bigcup_{e \in T'_E, \lambda(e) = t_i} e \tag{1}$$

Moreover, in order to preserve the time intervals in the selected subnet for the duplicated transitions, we use the λ function where any $e \in T'_E$, the minimum and the maximum waiting times are $\Delta(t) = [\delta_{min}(t), \delta_{max}(t)]$ where $\lambda(e) = t$. For example, we generate the Petri net in Figure 3 from the branching net of the time Petri net in Figure 2 using the assumption set $\mathscr{A} = \{(t_{RT}, 1)\}$ that leads to the selection of the subnet where there is only one *revise text* activity.

2.2 The RBAC Model

Different types of organizational structures give rise to different organizational models (Horling and Lesser, 2004). In most cases, the employees of the organization (i.e. its human resources) play one or more organizational roles according to their skills and characteristics. One of the most widely known models for capturing organizational structures based on roles is the Role-Based Access Control (RBAC) model (Colantonio et al., 2009).

Definition 5 (*RBAC Model*)
An *RBAC Model* is a 6-tuple $O = (A, R, L, S_{AL}, S_{RL}, S_{LL})$, where:

- A is the set of *activities* that corresponds to the activity transitions in a $TPN = (P, T, F, \nu, M_0, \Delta)$, hence $A \subseteq T$,
- R is the set of *resources*,
- L is the set of *roles*,

Table 1: RBAC model of the publishing entity

activity-to-role		*resource-to-role*		*role-to-role*	
Receive Manuscript	Publisher	Amy	Publisher	Publisher	Copy Editor
Proofread Manuscript	Copy Editor	Glen	Copy Editor		
Revise Text	Copy Editor	Drew	Copy Editor		
Revise Visual	Graphic Artist	Oliver	Graphic Artist		
Send Press Release	Administrative Assistant	Evan	Administrative Assistant		

- $S_{AL} \subseteq 2^{(A \times L)}$ is the set of activity-to-role assignment tuples specifying which activity can be executed by the resources associated with which role(s) (commonly known as *resource assignment* in BPM),
- $S_{RL} \subseteq 2^{(R \times L)}$ is the set of resource-to-role assignment tuples identifying the roles of a resource,
- $S_{LL} \subseteq 2^{(L \times L)}$ is the set of role-to-role assignment tuples that creates a hierarchical (sub-role) structure. The symbol $\preceq$ indicates the ordering operator. If $l1 \preceq l2$, then $l1$ is referred to as the *senior* of $l2$. Conversely, $l2$ is the *junior* of $l1$.

Let us assume that the book publishing process (cf. Figure 1 and Figure 2) is executed within an organization composed of five resources $R = \{Amy, Glen, Drew, Oliver, Evan\}$ that are assigned to four distinct roles $L = \{Publisher, Copy\ Editor, Graphic\ Artist, Administrative\ Assistant\}$. Table 1 shows a possible RBAC model of such a publishing entity. For instance, within the *activity-to-role* relation, the first entry means that the activity *Receive Manuscript* can be executed by the members of the role *Publisher*; within the *resource-to-role* relation, the first entry means that the resource *Amy* has the role *Publisher*; and within the *role-to-role* relation, the first entry means that the resources with the role *Publisher* can execute the same activities as the resources with the role *Copy Editor* (specifically, Publisher $\preceq$ Copy Editor).

After we describe *who* can execute *which* activity in a RBAC model, now we can define *which* activity requires *how many* resources to be performed.

Definition 6 (*Activity Resource Demand Set*)
The *activity resource demand set* $N_{AR} \subseteq 2^{(A \times \mathbb{Z}_{\geq 0})}$ represents the number of resources activity a demands to be executed.

Like the minimum and maximum duration of activities, the activity resource demand values are also estimated by process managers. In our running scenario, all activities demand only one resource except *Revise Text* which demands two resources.

2.3 Temporal Knowledge

In a real world scenario, the minimum and maximum duration of activities as well as the tentative deadlines for the completion of the process instances (i.e. each process execution) are estimated by the process managers. The process instances are usually managed with BPM Systems (BPMS) that provide functionality for defining and customizing process models, and for administrating and monitoring their executions. The execution data is stored in process execution (or event) logs or audit trails. The log entries comprise information about the events occurred during process execution, including resource-related properties (*who* executed each activity[3]) and temporal

[3] Within an organizational context, the resources that appear in the log are among those available according to the organizational model of the functional unit (e.g. an RBAC model).

Table 2: Example event log excerpt of the process in Figure 2 and the RBAC model in Table 1.

event id	trace id	event type	activity	resource	time stamp
e56	7	start	Receive Manuscript	Amy	2018-11-11T08:25:36
e57	6	start	Proofread Manuscript	Glen	2018-11-11T09:15:14
e58	7	end	Receive Manuscript	Amy	2018-11-11T09:21:44
e59	7	start	Proofread Manuscript	Drew	2018-11-11T09:35:51
e60	6	end	Proofread Manuscript	Glen	2018-11-11T10:26:14
e61	7	end	Proofread Manuscript	Drew	2018-11-11T12:45:10
e56	7	start	Receive Manuscript	Amy	2018-11-11T08:25:36

information (*when* each activity was executed). All the events related to a process instance constitute a *trace*.

Definition 7 (*Event Log*)
A *Event Log* is a 6-tuple $\mathscr{L} = (E, \varepsilon, \alpha, \rho, \tau, \mathscr{T})$, where:

- $E = \{e_1, e_2, ..., e_n\}$ is the set of *events*,
- $\varepsilon : E \to \{start, complete\}$ assigns the event type to events,
- $\rho : E \to R$ assigns the resources to events,
- $\alpha : E \to A$ assigns the activities to events,
- $\tau : E \to \mathbb{Z}_{\geq 0}$ assigns a timestamp to events,
- $\mathscr{T} = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ is the set of *traces* where $\sigma_i \in E^*$ is a trace such that each event appears only once and time is non-decreasing, i.e. $1 \leq j < k \leq |\sigma| : \sigma(j) \neq \sigma(k)$ and $\tau(\sigma(j)) \leq \tau(\sigma(k))$.

Table 2 shows an excerpt of an example event log. This retrospective data is crucial for the automated systems that support resource allocation. From such data together with organizational data (i.e. the RBAC model), using so-called process mining techniques we can identify the resource-activity and the role-activity relations necessary for resource allocation (Schönig et al., 2016). In addition, the *resource-activity* and *role-activity* durations can be estimated.

Definition 8 (*Resource-Activity Duration Function*)
the resource-activity duration function $\psi_{r,a} : (\mathscr{L} \times R \times A) \to \mathbb{Z}_{\geq 0}$ estimates the time it takes to execute an activity a by a resource r from the traces $\mathscr{T}$ in an event log $\mathscr{L}$

$$\psi_{r,a} = \frac{\sum_{\sigma_i \in \mathscr{T}} \sum_{(e_j, e_k) \in \Upsilon_{\sigma_i}} \tau(e_k) - \tau(e_j)}{\sum_{\sigma_i \in \mathscr{T}} \sum_{(e_j, e_k) \in \Upsilon_{\sigma_i}} 1} \quad \text{where} \tag{2}$$

$$\Upsilon_{\sigma_i} = \bigcup_{e_j \in \sigma_i, e_k \in \sigma_i} \begin{cases} (e_j, e_k) & \text{for } \rho(e_j) = \rho(e_k) = r,\ \alpha(e_j) = \alpha(e_k) = a,\ \varepsilon(e_j) = start,\ \varepsilon(e_k) = end \\ \emptyset & \text{otherwise.} \end{cases}$$

In short, $\psi_{r,a}$ calculates the mean resource-activity duration. For instance, assuming that the event log $\mathscr{L}$ in Table 2 contains traces $\mathscr{T}$ of the execution of the process in Figure 2 executed in an organization with the RBAC model depicted in Table 1, $\psi_{r,a}(\mathscr{L}, Glen, Proofread\ Manuscript)$ is computed as $1(hour)$ from the time difference between the start event `e57` and the end event `e60`.

Definition 9 (*Role-Activity Duration Function*)
The role-activity duration function $\psi_{l,a} : (\mathscr{L} \times O \times L \times A) \to \mathbb{Z}_{\geq 0}$ estimates the time it takes to a role l to execute an activity a from an event log $\mathscr{L}$ and an RBAC model O.

Table 3: Temporal knowledge

resource-activity duration		*role-activity duration*	
(Amy, Receive Manuscript)	1	(Publisher, Receive Manuscript)	1
(Drew, Proofread Manuscript)	3	(Copy Editor, Proofread Manuscript)	2
(Glen, Proofread Manuscript)	1	(Copy Editor, Revise Text)	6
(Glen, Revise Text)	6	(Graphic Artist, Revise Visual)	4
(Amy, Revise Text)	4	(Administrative Asst., Send Press Rel.)	3

$\psi_{l,a}$ is again of the form defined in (2), where in this case

$$\Upsilon_{\sigma_i} = \bigcup_{e_j \in \sigma_i, e_k \in \sigma_i} \begin{cases} (e_j, e_k) & \text{for } \rho(e_j) = \rho(e_k) = r,\ \alpha(e_j) = \alpha(e_k) = a,\ (r,l) \subseteq S_{RL},\ (a,l) \subseteq S_{AL}, \\ & \quad \varepsilon(e_j) = start,\ \varepsilon(e_k) = end. \\ \emptyset & \text{otherwise.} \end{cases} \tag{3}$$

In short, $\psi_{l,a}$ calculates the mean role-activity duration. For instance, assuming that the event log $\mathscr{L}$ in Table 2 contains traces of the execution of the process in Figure 2 executed in an organization with the RBAC model depicted in Table 1, $\psi_{l,a}(\mathscr{T}, Copy\ Editor, Proofread\ Manuscript)$ is computed as $2(hours)$ from the events `e57`, `e59`, `e60` and `e61`.

Table 3 shows possible temporal knowledge associated with our running example.

2.4 The Resource Allocation Problem in BPM

Resource allocation in BPM can be defined as follows. Let us consider a conflict-free (i.e. previously unfolded, if necessary) time Petri net $TPN = (P, T, F, M_0, \nu, \Delta)$ whose activities are in strict partial order, which means that the precedence relation between activities are irreflexive, transitive and asymmetrical. Each activity requires a number of (human) resources to be performed. The resources allowed to execute an activity are selected on the basis of the roles they are assigned in an organizational model (*activity-to-role* relation in the RBAC model) and hence, roles L are used to refer to resources in a process in a general way. For each role, per-period availability is constant and described in $S_{RL} \cup S_{LL}$.

Each activity a can have a different duration within a range. $\Delta(a) = [\delta_{min}(a), \delta_{max}(a)]$ specifies the minimum and maximum duration of an activity as estimated by the process manager. $\delta_{min}(a)$ is regarded as the *default duration* of a. Further temporal knowledge is extracted from an event log $\mathscr{L}$, namely *role-activity duration* $\psi_{(l,a)}$ and *resource-activity duration* $\psi_{(r,a)}$. The priority order of activity duration values is, assuming ">" is the *overwrites* relation, $\psi_{(r,a)} > \psi_{(l,a)} > \delta_{min}(a)$. Given a TPN, an event log $\mathscr{L}$, an RBAC model O and an upper bound on the process makespan u, all *derived resource-activity durations* are represented by the function $\delta_{(r,a)} : (R \times A) \to \mathbb{Z}_{\geq 0}$, where $A \subseteq T$ and $\delta_{(r,a)} \leq \delta_{max}(a)$.

For the formal representation of the resource allocation problem in BPM, we define a binary variable for each combination of a resource $r \in R$, an activity $a \in T$, and a starting time $s \in [0, u]$:

$$o_{ras} = \begin{cases} 1, & \text{if the resource } r \text{ is allocated to the activity } a \text{ and } a \text{ started at time } s, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Note that if $o_{ras} = 1$, the completion of an activity a occurs at time $s + \delta_{(r,a)}$. The objective function and the constraints of the model are as follows. For every $r_i, r_j \in R$, $a_m, a_n \in A$, $s_o, s_p \in U$, $U = \{0, 1, ..., u\}$, the objective is to minimize $max(C)$, where

$$C = \bigcup o_{r_i a_m s_o} \cdot (s_o + \delta_{(r_i, a_m)}) \tag{5}$$

so that:

$$\sum o_{r_i a_m s_o} = n_{a_m} \qquad (a_m, n_{a_m}) \in N_{AR} \tag{6}$$

$$o_{r_i a_m s_o} . o_{r_j a_m s_p} = 0 \qquad s_o \neq s_p \tag{7}$$

$$o_{r_i a_m s_o} . (s_o + \delta_{(r_i, a_m)}) \leq o_{r_j a_n s_p} . (s_p) \qquad a_m \rightarrow a_n \tag{8}$$

$$o_{r_i a_m s_o} . o_{r_i a_n s_p} = 0 \qquad a_m || a_n, [s_o, s_o + \delta_{(r_i, a_m)}] \cap [s_p, s_p + \delta_{(r_i, a_n)}] \neq \emptyset \tag{9}$$

Since the set of activities that are not followed by another activity are the last activities to be executed, the objective function (5) minimizes the completion time of the activity that has the greatest value. Constraint (6) indicates that an activity must be allocated as many resources as the activity demand function indicates (i.e. cardinality constraint). (7) ensures that no activity has more than one start time. (8) secures that no activity is started until all its predecessors are completed. Finally, (9) enforces that no resource is allocated to any parallel pair of activities that have overlapping execution periods.

A feasible allocation is a set of quadruples $I \subseteq 2^{(R \times A \times U \times U)}$ such that $(r, a, s_a, c_a) \in I$ satisfies the constraints (6-9) where each activity $a \in A$ is assigned a resource $r \in R$, a start time $s_a \in U$ and a completion time $c_a = s_a + \delta_{(r,a)}$.

Our benchmark problem has similar characteristics to the Multi-mode Resource-constrained Project Scheduling Problem (MRCPSP) (Sprecher et al., 1994). Both problems have multiple sets of renewable resources[4], different resource demands per activity, precedence constraints of activities defined by a directed acyclic graph, and an objective function to optimize the makespan. The main difference between the two methods is that our problem has a more general temporal setting involving activity durations defined by *defaults* which can be overwritten by more specific resource-activity durations (in contrast to MRCPSP's static modes); and (ii) our problem has a simpler resource setting as we do not consider the nonrenewable resources[5].

As for complexity, our problem is a generalization of job shop scheduling (JSP) problem which is known to be NP-Complete (Garey et al., 1976) for its variants with two or more machines and with makespan optimization. Trivially, this variant can be solved as a resource allocation problem where the input Petri net has as many parallel branches as the number of jobs in the JSP problem, the ordered tasks of each job map to activities in direct precedence relation with each other in separate branches, machines map to roles with only one resource, and the machine duration per task is represented as role-activity duration. Consequently, the computational complexity of the decision variant of our problem (i.e., is there a feasible allocation at makespan k) is NP-Complete and the optimization variant of our problem (i.e., is there an optimal allocation at minimized makespan k_{min}) is NP-Hard. In this paper we provide an encoding of resource scheduling in ASP with weak constraints (and without disjunction), which captures the class of FP^{NP}, that typically is used for computing optimal solutions of such NP-complete problems with a given bound k (Buccafurri et al., 1997). Therefore, our resource allocation problem seems to be an ideal candidate for benchmarking ASP systems supporting weak constraints.

[4] Renewable resources are available with a constant amount in each time period.
[5] The availability of the nonrenewable resources is limited for the whole time horizon of the project.

3 Resource Allocation in Business Processes with ASP

For benchmarking the ASP systems, we have encoded the problem of resource allocation in business processes as an ASP program and we have developed a problem instance generator. Before describing how we have done that in Sections 3.2 and 3.3, respectively, in Section 3.1 we introduce fundamental concepts in ASP.

3.1 Fundamentals of Answer Set Programming (ASP)

ASP (Brewka et al., 2011) is a declarative (logic-programming-style) paradigm for solving combinatorial search problems. An *ASP program* Π is a finite set of rules of the form

$$a_0 \texttt{:-} a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n. \tag{10}$$

where $n \geq m \geq 0$ and each a_i is a function-free first-order atom; if $n = m = 0$ we call r a *fact*. "*not*" is called *negation as failure*. Different from the classical negation (i.e., $\neg a$), *not a* is derived from failure to derive a. Sets of rules are evaluated in ASP under the so-called *stable-model semantics*, which allows several models (so called *answer sets*). We again refer to (Brewka et al., 2011) and references therein for details.

Whenever a_i is a first-order predicate with variables within a rule of the form (10), this rule is considered a shortcut for its "grounding" $ground(r)$ (i.e. the set of its ground instantiations obtained by replacing the variables with all possible constants occurring in Π). Likewise, we denote by $ground(\Pi)$ the set of rules obtained from grounding all rules in Π.

If a_0 is empty in a rule r, we call r a *constraint* which rules out models satisfying its body atoms. It is used to eliminate unwanted solution candidates. As a syntactic extension, ASP Core-2 standard (Calimeri et al., 2013) allows set-like *choice expressions* of the form $x \leq \{a_1, \ldots, a_m\} \leq y \texttt{:-} a_{m+1}, \ldots, a_l$, that is, if the body holds then to choose arbitrarily a subset of $\{a_1, \ldots, a_m\}$ of size x to y to include in a model.

Optimization (i.e., minimization or maximization) statements indicate preferences between possible answer sets:

$$\texttt{:}\sim\ a_1, \ldots, a_m.[w, t_1, \ldots, t_n]$$

where $t_1, \ldots, t_n$ are terms (e.g., atoms), and w is weight. The answer sets of a program Π plus optimization statements are the answer sets of Π with least violation of these statements (i.e., the sum of weights w over all occurrences of weighted atoms that are satisfied by a stable model) called best models.

Another extension we use is the aggregates. Aggregates are arithmetic operations over a set of elements and they occur in aggregate atoms in rule bodies that have the form $\#aggr\{w_1 : a_1, \ldots, w_n : a_n\} \prec u$, where w_i is weight assigned to a_i, the operation $aggr \in \{$"`count`","`sum`", "`min`","`max`"$\}$, the relation $\prec \in \{" < ", " \leq ", " = ", " \neq ", " > ", " \geq "\}$, and u is a ground term. For instance, $\#\texttt{sum}\{w_1 : a_1, \ldots, w_n : a_n\} \prec u$ is *true* if $\sum_{i=1}^{n} w_i \prec u$, and $false$ otherwise. `#count` is basically `#sum` where all the weights set to one.

ASP systems typically first compute (a subset of) $ground(\Pi)$, and then use a Davis-Putnam-Logemann-Loveland- (DPLL)-like branch and bound algorithm to find answer sets for this ground program. There are various *grounders* and *solvers* for ASP problem specifications (Gebser et al., 2017b). We will use two state-of-the-art ASP grounders and solvers, GRINGO (Gebser et al., 2011) and I-DLV (Calimeri et al., 2017), and CLASP (Gebser et al., 2015) and WASP (Alviano

```
dPrec(T,T1) :- iPlace(P,T), oPlace(P1,T), iPlace(P1,T1), P!=P1, T!=T1.                    1

naPrec(T,T1) :- dPrec(T,T1), not aTransition(T), not aTransition(T1).                     2
naPrec(T,T2) :- nadPrec(T,T1), nadPrec(T1,T2).                                            3

adPrec(A,A1) :- dPrec(A,A1), aTransition(A), aTransition(A1).                             4
adPrec(A,A1) :- dPrec(A,T1), dPrec(T1,A1), aTransition(A), aTransition(A1), not           5
    aTransition(T1).
adPrec(A,A1) :- dPrec(A,T), naPrec(T,T1), dPrec(T1,A1), aTransition(A),                   6
    aTransition(A1).

aPrec(A,A1) :- adPrec(A,A1).                                                              7
aPrec(A,A1) :- adPrec(A,A2), adPrec(A2,A1).                                               8

aPar(A,A1) :- not aPrec(A,A1), aTransition(A), aTransition(A1).                           9
aPar(A,A1) :- aPar(A1,A).                                                                 10
```

Fig. 4: ASP encoding to preprocess the Petri net to obtain the preceding and parallel activities

et al., 2015) respectively, for testing them against our benchmark (cf. Section 4), as they have among the most efficient implementations (Gebser et al., 2017b).

3.2 ASP Encoding

Our encoding is shown in Figures 4 and 5. The assumptions that we make about the structure of a resource allocation problem are as follows: (i) There is no preemption (i.e. each activity, once started, must be completed without interruptions); and (ii) the duration of the activities is independent of the schedule, and it is known in advance. These assumptions are common in related approaches (Lombardi and Milano, 2012).

Input Format. The input is divided into three groups of predicates as follows.
A Time Petri net $TPN = (P,T,F,M_0,\nu,\Delta)$*:*
`iPlace(`p`,`t`)`: place $p \in P$ is an input place of activity transition $t \in T$, $(p,t) \in F$
`oPlace(`p`,`t`)`: place $p \in P$ is an output place of activity transition $t \in T$, i.e. $(t,p) \in F$
`aTransition(`a`)`: $a \in T$ is an activity transition
`minActDuration(`a`,`d`)`: the minimum waiting time/default duration of activity a is $\delta_{min}(a) = d$
`maxActDuration(`a`,`d`)`: the maximum waiting time of activity a is $\delta_{max}(a) = d$

Temporal knowledge extracted from the event log $\mathscr{L}$*:*
`raDuration(`r`,`a`,`d`)`: duration of activity a is d when it is executed by resource r (i.e. $\psi_{(r,a)} = d$)
`laDuration(`l`,`a`,`d`)`: duration of activity a is d by a resource that has role l (i.e. $\psi_{(l,a)} = d$)
`upperBound(`u`)`: makespan is bounded at u time units (TU)

RBAC model $O = (A,R,L,S_{AL},S_{RL},S_{LL})$*:*
`alAC(`a`,`l`)`: resources with role l can execute activity a (i.e. $(a,l) \in S_{AL}$)
`rlAC(`r`,`l`)`: resource r has role l (i.e. $(r,l) \in S_{RL}$)
`llAC(`l_1`,`l_2`)`: resources with role l_1 can execute the same activities that the resources with role l_2 can execute (i.e. $(l_1,l_2) \in S_{LL}$)
`aDemand(`a`,`n`)`: an activity a requires n many resources to be executed, (i.e. $\chi(a) = n$)

Output Format. For the output we use the following predicate:
`allocate(`r`,`a`,`s`,`c`)`: a resource r is allocated to activity a at the start time s until the completion time c, i.e. $(r,a,s,c) \in I$

```
time(0).                                                                                    11
time(T1) :- T1=T+1, time(T), T1<=U, upperBound(U).                                          12

alAC(A,L1) :- llAC(L1,L2), alAC(A,L2).                                                      13

raDuration2(R,A,D1) :- maxActDuration(A,D), raDuration(R,A,D1), D1<=D.                      14
laDuration2(R,A,D1) :- maxActDuration(A,D), laDuration(R,A,D1), D1<=D.                      15

defaultRAD(R,A,D) :- raDuration2(R,A,D), rlAC(R,L), alAC(A,L).                              16
defaultRAD(R,A,D) :- not raDuration2(R,A,D1), laDuration2(L,A,D), rlAC(R,L),                17
     alAC(A,L), time(D1).
defaultRAD(R,A,D) :- not raDuration2(R,A,D1), not laDuration2(L,A,D2),                      18
     minActDuration(A,D), rlAC(R,L), alAC(A,L), time(D1), time(D2).

defaultRAD(R,A,D1) :- defaultRAD(R,A,D), defaultRAD(R1,A,D1), R!=R1, D1>D.                  19

N<={allocate(R,A,S,E): time(S), defaultRAD(R,A,D), E=S+D}<=N :- aTransition(A),             20
     aDemand(A,N).

:- allocate(R,A,S,E), upperBound(U), E>U.                                                   21

:- allocate(R,A,S,E), allocate(R,A,S1,E1), S<S1.                                            22
:- allocate(R,A,S,E), allocate(R,A,S1,E1), E<E1.                                            23

:- allocate(R,A,S,E), allocate(R1,A,S1,E1), R1!=R, S<S1.                                    24
:- allocate(R,A,S,E), allocate(R1,A,S1,E1), R1!=R, E<E1.                                    25

:- adPrec(A,A1), allocate(R,A,S,E), allocate(R1,A1,S1,E1), E>S1.                            26

:- aPar(A,A1), allocate(R,A,S,E), allocate(R,A1,S1,E1), S<E1, S>=S1, A<A1.                  27
:- aPar(A,A1), allocate(R,A,S,E), allocate(R,A1,S1,E1), E>S1, E<=E1, A<A1.                  28
:- aPar(A,A1), allocate(R,A,S,E), allocate(R,A1,S1,E1), S<=S1, E>E1, A<A1.                  29
:- aPar(A,A1), allocate(R,A,S,E), allocate(R,A1,S1,E1), S1<=S, E1>E, A<A1.                  30

greaterExists(E) :- allocate(R,A,S,E), allocate(R1,A1,S1,E1), E1>E.                         31
maxC(E) :- not greaterExists(E), allocate(R,A,S,E).                                         32

:~ maxC(E). [E@1,E]                                                                         33
```

Fig. 5: ASP encoding for the resource allocation problem

3.2.1 Code Description

We first obtain the ordering relations of activities *precedence* and *parallel* (cf. Section 2) using the encoding in Figure 4. Rules (1-6) identify the activities that are in direct precedence (e.g. `adPrec`(a_1,a_2) means that $a_1 \xrightarrow{a} a_2$). Rules (7,10) identify the activities that can be executed in parallel (e.g. `aPar`(a_1,a_2) means that $a_1||a_2$) in a smart way by considering the activity-to-role RBAC relations given in the form `alAC`(l,a).

Figure 5 is our ASP program for allocating resources to process activities. Rules (11,12) generate the finite time domain using the upper bound. Rule (13) propagates the permissions of activity executions of a role $l2$ to another role $l1$ (i.e. role-to-role RBAC relations). Rules (14-19) handle the preference $(>)$ of activity durations: *resource-activity duration* $>$ *role-activity duration* $>$ *default activity duration*. Rule (20) generates the allocations for each activity (i.e. activity resource demand is enforced, (6) in Section Section 2). Constraint (21) restricts the upper bound on the completion time. Constraints (22,23) enforce a resource to be allocated to an activity only once and constraints (24,25) impose different resources that are allocated to an activity to start and end their execution at the same time (i.e. (7) in Section 2). Constraint (26) enforces the time constraints on directly preceding activities (i.e. (8) in Section 2). Constraints (27-30) enforce the time and resource constraints on parallel activities (e.g. two parallel activities cannot be executed by the same resource at the same time, i.e. (9) in Section 2). Rules (31,32) compute the maximum completion time (i.e the makespan). Weak constraint (33) optimizes the makespan.

Table 4: Parameters for generating resource allocation problem instances

Petri net generator		*RBAC generator*		*Temporal knowledge generator*	
Number of activities	n_A	Number of resources	n_R	Upper bound	u
Degree of parallelism	δ_p	Number of roles	n_L	Number of resource-activity duration	$n_{d_{RA}}$
				Number of role-activity duration	$n_{d_{LA}}$

3.2.2 Running Example

The result of the running example described in Section 2 is computed by our program as follows:

```
allocate(amy,tRM,0,1) allocate(glen,tPM,1,2)   allocate(glen,tRT,2,8)
allocate(amy,tRT,2,6) allocate(oliver,tRV,2,6) allocate(glen,tPM2,8,9)
allocate(evan,tSPR,9,12)
```

The optimized makespan is 12 hours. As *Receive Manuscript* can only be performed by the role *Publisher*, and therefore Amy is allocated to it. Glen is allocated to *Proofread Manuscript* due to the fact that the solver knows he can execute it faster than others (cf. `raDuration(`*glen*`,`*aPR*`,1)`). Glen and Amy are allocated to *Revise Text* as it requires two resources to be performed (cf. `aDemand(`*aRT*`,2)`). *Revise Visual* takes 4 hours because `laDuration(`*graphicArtist*`,`*aRV*`,4)` overwrites the default duration `minActDuration(`*aRV*`,5)`.

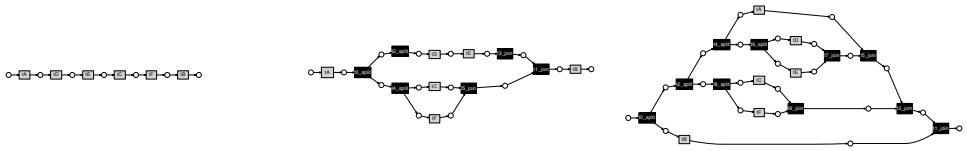
3.3 Problem Instance Generator

A formalized problem instance generator is necessary for a thorough evaluation of ASP systems against problems that have different sizes and properties. Our problem instance generator consists of three main parts: a Petri net generator; an RBAC model and activity resource demand generator; and a temporal knowledge generator. The required parameters for each of them are provided in Table 4.

Petri net generator: For our benchmark we first generate conflict-free Petri nets using the *Generate block-structured stochastic Petri net* plug-in (Rogge-Solti, 2014) of the process mining tool ProM (van der Aalst, 2016). This generator performs a series of random structured insertion operations of new control-flow constructs resulting in a random Petri net that is by generation sound, free-choice and block-structured. The input of the Petri net generator is as follows: the number of activities $n_A \in \mathbb{Z}^+$ and the degree of parallelism of the generated net $\delta_p \in \mathbb{Z}_{\geq 0}$, and $0 \leq \delta_p \leq 100$. The generated Petri net $PN = (P, T, F, M_0, \nu)$ is described with the predicates: `place(`p`)` (i.e. $p \in P$), `transition(`t`)` (i.e. $t \in T$), `aTransition(`a`)` (i.e. $a \in A$ and $A \subseteq T$), `iPlace(`p`,`t`)` (i.e. $p \in {}^\bullet t$) and `oPlace(`p`,`t`)` (i.e. $p \in t^\bullet$). By changing the values of the input parameters, different sequentiality and parallelism degrees are realized in the generated Petri nets. For instance, in Figure 6 where $n_A = 6$, (a) has no parallel activities, (c) has no sequential activities, and (b) has both parallel and sequential activities.

The generated Petri nets could be understood as being equivalent to the conflict-free Petri net that would arise from translating, e.g., a BPMN model to a Petri net first; and then deriving the branching net of this net and selecting a subnet that holds an assumption set.

As the RBAC model requires the set of activities A, which is necessary for activity-to-role assignments, now we can further continue describing RBAC model generation.



(a) $\delta_p = 0$ (b) $\delta_p = 50$ (c) $\delta_p = 100$

Fig. 6: Three generated Petri nets

RBAC model and activity resource demand generator: We generate a RBAC model $O = (A, R, L, S_{AL}, S_{RL}, S_{LL})$ where $A \subseteq T$ given a number of resources n_R, a number of roles n_L and the output of the Petri net generator by using the following ASP program:

Input: `aTransition/1, nR/1, nL/1`

Output: `alAC/2, rlAC/2, llAC/2, aDemand/2`

```
resource(1). role(1).                                                          34
resource(R1) :- resource(R), nR(N), R1=R+1, R1<=N.                             35
role(L1) :- role(L), nL(N), L1=L+1, L1<=N.                                     36

1<={alAC(A,L):role(L)} :- aTransition(A).                                      37
1<={rlAC(R,L):role(L)} :- resource(R).                                         38
{llAC(L,L1):role(L1)} :- role(L).                                              39
:- llAC(L,L).                                                                  40

nrlAC(L,N):- N=#count{R:rlAC(R,L)}, role(L).                                   41
{aDemand(A,N)}<=N :- aTransition(A), alAC(A,L), nrlAC(L,N).                    42
```

Lines (34-36) generate $R = \{r_1, r_2, ..., r_{n_R}\}$ and $L = \{l_1, l_2, ..., l_{n_L}\}$ using the parameters n_R and n_L. Line (37) generates the set S_{AL} (i.e. every activity a has at least one assigned role l). Line (38) generates the set S_{RL} (i.e. every resource r has at least one assigned role l). Line (39) generates the set S_{LL} and constraint (40) guarantees that a role l cannot be senior to itself. Lines (41,42) generate the activity resource demand set (i.e. N_{AR}).

Temporal knowledge generator: We generate the temporal knowledge (i.e. $\psi_{(r,a)}$ and $\psi_{(l,a)}$) along with the default durations of activities (i.e. for each activity a, $\delta_{min}(a)$) given an upper bound u, a number of resource-activity duration $n_{d_{RA}}$, a number of role-activity duration $n_{d_{LA}}$ and the output from the previously defined generators by using the following ASP program:

Input: `upperBound/1, ndRA/1, ndLA/1, aTransition/1, alAC/2, rlAC/2, llAC/2`

Output: `minActDuration/1, raDuration/2, laDuration/2`

```
nA(N) :- #count{A:activity(A)}=N.                                              43
maxActivityDuration(U/N) :- upperBound(U), nA(N).                              44
aDurationDomain(M) :- maxActivityDuration(M).                                  45
aDurationDomain(M1) :- aDurationDomain(M), M>0, M1=M-1.                        46

1<={minActDuration(A,D): aDurationDomain(D)}<=1 :- aTransition(A).             47

N<={raDuration(R,A,D): alAC(A,L), rlAC(R,L), aDurationDomain(D)}<=N :- ndRA(N). 48
:- raDuration(R,A,D),raDuration(R,A,D1),D!=D1.                                 49

alAC(A,L1) :- llAC(L1,L2), alAC(A,L2).                                         50
N<={laDuration(L,A,D): alAC(A,L), aDurationDomain(D)}<=N :- ndLA(N).           51
:- laDuration(L,A,D),laDuration(L,A,D1),D!=D1.                                 52
```

Lines (43-46) estimates the maximum value of activity duration from the upper bound u and derives a variable domain for the activity durations. Line (47) assigns one default duration per activity (i.e. $\delta_{min}(a)$). Lines (48,49) generates the resource-activity durations. Lines (50-52) generates the role-activity durations.

Table 5: Properties of problem instances

id	n_A	δ_p	n_R	n_L	*SAT*	u	id	n_A	δ_p	n_R	n_L	*SAT*	u	id	n_A	δ_p	n_R	n_L	*SAT*	u
1	8	50	2	1	*yes*	90	**24**	16	30	16	4	*yes*	90	**47**	32	90	16	8	*yes*	330
2	8	75	4	1	*yes*	70	**25**	16	30	16	4	*yes*	85	**48**	32	80	16	8	*yes*	260
3	8	100	4	1	*yes*	105	**26**	16	90	16	4	*yes*	140	**49**	32	90	16	8	*yes*	310
4	8	100	8	2	*yes*	70	**27**	16	30	37	8	*yes*	65	**50**	32	90	16	8	*yes*	360
5	16	90	2	1	*yes*	200	**28**	16	50	35	8	*yes*	75	**51**	32	50	17	4	*yes*	145
6	16	50	4	1	*yes*	75	**29**	16	75	35	8	*yes*	130	**52**	32	50	17	4	*yes*	145
7	16	90	8	4	*yes*	175	**30**	16	90	34	8	*yes*	190	**53**	32	50	32	16	*yes*	160
8	16	50	4	1	*yes*	120	**31**	32	90	2	1	*yes*	330	**54**	32	80	16	4	*yes*	330
9	16	90	4	1	*yes*	185	**32**	32	95	2	1	*yes*	360	**55**	32	50	33	8	*yes*	380
10	16	75	4	1	*yes*	145	**33**	32	30	4	1	*yes*	100	**56**	32	90	32	16	*yes*	345
11	16	90	4	1	*yes*	210	**34**	32	60	4	1	*yes*	210	**57**	32	90	17	4	*yes*	350
12	16	90	9	2	*yes*	205	**35**	32	75	8	4	*no*	210	**58**	32	50	32	16	*yes*	235
13	16	50	8	2	*yes*	130	**36**	32	60	4	1	*yes*	275	**59**	32	50	16	4	*yes*	360
14	16	75	8	2	*yes*	75	**37**	32	30	4	1	*yes*	270	**60**	32	80	32	16	*yes*	290
15	16	40	16	8	*yes*	85	**38**	32	85	4	1	*yes*	270	**61**	32	60	32	16	*yes*	520
16	16	40	8	2	*yes*	75	**39**	32	85	4	1	*yes*	300	**62**	32	50	33	8	*yes*	150
17	16	90	16	8	*yes*	190	**40**	32	90	4	1	*yes*	145	**63**	32	80	34	8	*yes*	317
18	16	90	8	2	*yes*	210	**41**	32	90	8	4	*yes*	355	**64**	32	90	33	8	*yes*	340
19	16	75	8	2	*yes*	115	**42**	32	90	8	4	*yes*	360	**65**	64	90	8	4	*no*	305
20	16	75	16	8	*yes*	120	**43**	32	30	8	2	*yes*	200	**66**	64	90	8	4	*no*	310
21	16	90	16	8	*yes*	165	**44**	32	75	16	8	*yes*	280	**67**	64	60	16	8	*yes*	295
22	16	90	9	2	*yes*	200	**45**	32	90	8	2	*yes*	315	**68**	64	90	16	8	*yes*	710
23	16	30	17	4	*yes*	75	**46**	32	50	16	8	*yes*	170	**69**	64	60	64	32	*yes*	320
														70	64	90	64	32	*yes*	620

4 Benchmark for ASP Systems

Our benchmark is configurable in the sense that any combinations of given ASP grounders and ASP solvers are compared against each other. In order to get a good idea of how challenging our problem is for the state-of-the-art ASP systems, we have selected the ASP *grounders* GRINGO (Gebser et al., 2011) and I-DLV (Calimeri et al., 2017), as well as the ASP *solvers* CLASP (Gebser et al., 2015) and WASP (Alviano et al., 2017). We have selected those grounders and solvers due to their top rankings in the latest ASP Competition (Gebser et al., 2017b). Every possible grounder+solver combination is competed against each other. Hence, our baseline benchmark run consists of four different ASP grounder+solver configurations: GRINGO+CLASP (i.e. CLINGO (Gebser et al., 2014)), GRINGO+WASP, I-DLV+CLASP, and I-DLV+WASP (i.e. DLV 2.0 (Alviano et al., 2017)). We use the most up-to-date versions of these grounders and solvers: GRINGO 5.3.0, I-DLV 1.1.1, CLASP 3.3.4, WASP 2.0. These tools are executed on their default options (i.e., no parameter is given).

Platform: The benchmark has been run on an Ubuntu Linux server (64bit), equipped with 16 core 2.40 GHz Intel Xeon Processor and 128 GB RAM. Time and memory for each run were limited to 2 hours CPU clock time and 20 GB, respectively.

Problem Instances: We have generated 70 problem instances using our instance generator. The details of these instances are provided in Table 5. In the table, **id** is the unique identifier of each instance, n_A is the number of activities to which resources are going to be allocated, δ_p is the degree of parallelism of the generated Petri net, n_R is the number of resources, and n_L is the number of roles. The *SAT* column indicates if there exists a feasible solution for the problem instance, in other words, if there is a feasible makespan c_{max} where $c_{max} \leq u$. u is the upper bound for the makespan for the solutions as described in Section 3.3.

Benchmark Results: We have summarized the benchmark results in Tables 6 and 7. Table 6 shows the performance statistics of two grounders while grounding the problem instances. *time* is the CPU time in seconds, *mem* is the memory used in MB, and $|g(\Pi)|$ is the size of the ground program in MB. Table 7 presents the performance statistics of only the ASP solvers. CLASP(GRINGO) indicates the performance of the solver CLASP using the ground program from the grounder GRINGO. c_{max} is the makespan computed by the system, which is minimized by

Table 6: Grounder statistics

	GRINGO			I-DLV				GRINGO			I-DLV		
id	*time*	*mem*	$\|g(\Pi)\|$	*time*	*mem*	$\|g(\Pi)\|$	**id**	*time*	*mem*	$\|g(\Pi)\|$	*time*	*mem*	$\|g(\Pi)\|$
1	3	7	29	3	74	20	**36**	1689	22	16323	960	4403	4526
2	7	8	74	4	86	25	**37**	1572	23	15544	1049	4272	4347
3	19	8	163	11	174	56	**38**	1459	18	15766	855	4284	4363
4	7	7	85	3	129	28	**39**	1860	20	19515	1107	5158	5422
5	68	9	527	76	598	330	**40**	460	15	4465	265	1300	1218
6	30	9	316	15	193	91	**41**	672	15	6743	1176	8736	3818
7	30	9	287	52	737	188	**42**	445	15	6939	925	8969	3927
8	77	9	752	46	439	239	**43**	513	18	8369	491	3552	2376
9	192	10	1858	116	1024	570	**44**	233	16	4192	26	150	292
10	112	10	1129	66	643	350	**45**	1245	24	21315	1307	8285	6056
11	222	11	2301	137	1226	700	**46**	94	11	1519	10	92	115
12	291	13	2977	252	2003	809	**47**	356	15	5810	37	174	394
13	95	11	972	87	821	311	**48**	205	15	3593	23	142	252
14	29	9	297	5	179	46	**49**	291	14	5129	31	164	350
15	8	8	97	1	30	14	**50**	463	18	6912	44	187	465
16	27	9	307	5	180	46	**51**	290	13	5130	20	96	182
17	43	10	456	6	54	67	**52**	309	16	5530	21	97	191
18	259	11	2449	227	1870	751	**53**	78	11	1334	8	141	111
19	75	9	684	65	595	222	**54**	1283	25	23428	2266	14761	6651
20	21	8	184	3	39	27	**55**	2011	28	32896	120	373	1073
21	31	9	344	4	49	49	**56**	374	17	6341	35	293	447
22	242	11	2817	151	1869	735	**57**	1724	21	29670	1508	18263	8335
23	37	8	345	3	32	25	**58**	154	12	2900	16	207	217
24	45	9	438	3	34	32	**59**	1643	27	27747	1431	18011	8234
25	40	8	398	3	33	29	**60**	253	16	4495	25	251	325
26	87	10	1025	17	567	157	**61**	807	19	14386	83	456	993
27	29	9	308	2	42	26	**62**	282	13	5158	20	154	201
28	32	9	351	3	45	30	**63**	1461	26	24932	85	325	800
29	109	10	1161	8	74	84	**64**	1624	26	26446	102	335	871
30	244	11	2214	17	96	148	**65**	1120	26	19233	93	198	726
31	591	14	6122	489	3151	3320	**66**	1268	59	20685	114	204	768
32	814	15	6860	776	3688	3744	**67**	1094	26	18091	86	308	678
33	246	12	2278	159	717	638	**68**	-	37	93938	512	737	3847
34	1092	15	9483	669	2598	2582	**69**	1242	22	21228	78	981	925
35	232	12	2199	21	72	161	**70**	5118	45	80788	279	1907	3130

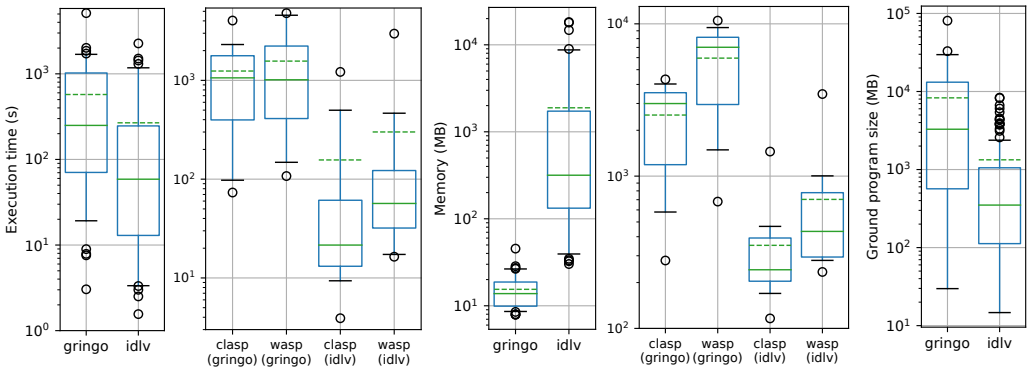
the weak constraint. Under this column, bold and italic numbers are the confirmed optima cases; other values are the instances for which a solution is found but the solution is not proven to be the optimal one (e.g. when the time or the memory limit is reached). For both of the tables "–" under *time* column means "out-of-time" and "–" under *mem* column means "out-of-memory".

The box plots in Figure 7 visually summarize Tables 6 and 7. We logarithmically scaled the y-axis of the plots for the sake of readability. 90% of the samples are in between the upper and lower whiskers. The solid green line represents the *median*, and the dashed green line represents the *mean* of the sample. Figure 7(a) and Figure 7(b) compare the CPU execution times of grounders and solvers. Fig. 7(c) and Figure 7(d) compare the memory usage of grounders and solvers. Figure 7(e) reports on the ground program sizes of the instances. By looking at Figure 7(a) and Figure 7(c), we find out that I-DLV grounds the problem instances much faster than GRINGO, however it has a larger memory footprint. The program rewriting (i.e. intelligent projections) feature of I-DLV is a vital optimization for this problem as the ground programs that are generated by I-DLV are way smaller than those generated by GRINGO (cf. Table 6 and Figure 7(e)).

The cactus plots in Figure 8 separately compare the grounder and solver performances. Less steep curves (cf. Figure 8(b)) and smaller memory footprint (cf. Figure 7(d)) of CLASP(GRINGO) and CLASP(I-DLV)–in comparison to those of WASP(GRINGO) and WASP(I-DLV)–concludes that the ASP solver CLASP performs better than WASP for our benchmark problem.

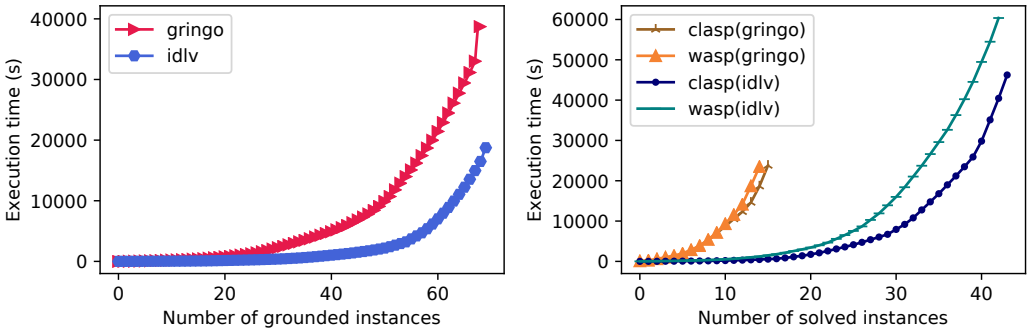
Table 7: Solver statistics

		CLASP(GRINGO)			WASP(GRINGO)			CLASP(I-DLV)			WASP(I-DLV)		
id	u	*time*	*mem*	c_{max}	*time*	*mem*	c_{max}	*time*	*mem*	c_{max}	*time*	*mem*	c_{max}
1	90	315	279	***69***	533	680	***69***	345	169	***69***	465	415	***69***
2	70	73	581	***54***	107	1487	***54***	12	200	***54***	37	485	***54***
3	105	720	1426	***79***	1535	3588	***79***	498	439	***79***	415	855	***79***
4	70	97	733	***55***	371	1875	***55***	21	257	***55***	50	702	***55***
5	200	–	5964	200	–	14000	192	3923	1865	***149***	–	3639	179
6	75	–	2947	68	–	7220	68	–	543	68	3037	1109	***68***
7	175	2305	3360	***133***	4788	7821	***133***	1217	1449	***133***	2971	3459	***133***
8	120	–	7912	98	–	18600	101	–	1302	92	–	2650	92
9	185	5302	–		192	19007		–	3196		–	6245	176
10	145	7146	12026		106	–		5279	1941	***108***	5014	3908	***108***
11	210	6646	–		207	–		–	3876		–	7567	
12	205	–	11788		315	–		–	5156	192	–	10920	
13	130	–	10487	134	96	–		2025	2042	***104***	2084	4368	***104***
14	75	1785	3190	***59***	2278	7411	***59***	84	467	***59***	133	983	***59***
15	85	115	951	***64***	148	2317	***64***	3	116	***64***	17	234	***64***
16	75	935	2985	***56***	432	6962	***56***	34	456	***56***	56	1003	***56***
17	190	5134	5547	***144***	7157	12500	160	68	432	***144***	279	418	***144***
18	210	–	10295		267	–		–	4825	206	–	10081	
19	115	–	8690	109	–	18098	100	377	1482	***86***	929	3162	***86***
20	120	479	2067	***92***	889	4860	***92***	13	205	***92***	33	279	***92***
21	165	1739	4026	***124***	4562	9450	***124***	37	344	***124***	111	373	***124***
22	200	–	11283		253	–		–	4753	160	–	9817	195
23	75	1766	3395	***64***	1730	8062	***64***	10	202	***64***	25	283	***64***
24	90	4032	4319	***64***	2545	10498	***64***	25	234	***64***	57	299	***64***
25	85	1785	3793	***67***	2171	9018	***67***	14	217	***67***	30	289	***67***
26	140	–	11780		83	–		599	1571	***106***	1609	2888	***106***
27	65	1060	2995	***48***	388	7016	***48***	9	242	***48***	16	433	***48***
28	75	1433	3664	***56***	1013	8255	***56***	17	260	***56***	83	449	***56***
29	130	–	12586	122	101	–		91	609	***99***	183	803	***99***
30	190	–	9359		260	–		650	1026	***143***	933	1082	***143***
31	330	621	–		1010	–		–	11872		284	–	
32	360	670	–		1168	–		–	13344		297	–	
33	100	–	10537		258	–		–	2467		–	5751	
34	210	635	–		1006	–		–	9483		287	–	
35	210	5856	–		234	–		271	919		411	553	
36	275	605	–		928	–		–	16237		312	–	
37	270	553	–		938	–		–	15679	145	272	–	
38	270	582	–		869	–		–	15746		243	–	
39	300	635	–		1070	–		997	–		221	–	
40	145	618	–		555	–		–	4612		–	10675	
41	355	634	–		1017	–		796	–		161	–	
42	360	406	–		795	–		565	–		145	–	
43	200	366	–		628	–		–	10440	76	170	–	
44	280	–	17891		396	–		545	1770	***210***	1119	1258	***210***
45	315	366	–		660	–		803	–		124	–	
46	170	–	17411		87	–		57	769	***125***	170	701	***125***
47	330	414	–		793	–		2194	2289	***250***	3650	2063	***250***
48	260	–	14796		307	–		499	1559	***197***	1034	1079	***197***
49	310	386	–		573	–		1241	2069	***233***	2896	1824	***233***
50	360	391	–		786	–		1998	2818	***271***	4946	2654	***271***
51	145	356	–		450	–		197	1142	***110***	288	954	***110***
52	145	347	–		506	–		469	1194	***118***	666	1017	***118***
53	160	–	15506	150	94	–		58	846	***122***	204	1076	***122***
54	330	414	–		757	–		790	–		132	–	
55	380	397	–		755	–		2286	6601	***163***	2644	4370	***163***
56	345	415	–		723	–		1975	2953	***258***	4291	2695	***258***
57	350	369	–		672	–		456	–		47	–	
58	235	6300	–		183	–		102	1522	***132***	553	1580	***132***
59	360	384	–		770	–		465	–		48	–	
60	290	352	–		448	–		655	2120	***215***	1622	1962	***215***
61	520	399	–		812	–		677	5929	***123***	2685	3665	***123***
62	150	357	–		479	–		266	1437	***126***	366	1631	***126***
63	317	394	–		769	–		5339	4903	***238***	5881	4326	***238***
64	340	412	–		769	–		–	5619	313	–	4394	274
65	305	414	–		783	–		1640	3865		2002	1933	
66	310	396	–		713	–		2072	4057		2397	2091	
67	295	416	–		769	–		2424	3908	***221***	3945	2545	***221***
68	710							481	–		–	7885	
69	320	395	–		760	–		5794	6307	***238***	–	7863	313
70	620	378	–		838	–		7152	–		–	16188	



(a) (b) (c) (d) (e)

Fig. 7: Box plots regarding the performance statistics of grounders and solvers



(a) Grounding times (b) Solving times

Fig. 8: Sorted cactus plots

Overall, I-DLV+CLASP completes 41 instances in given time and memory constraints whereas I-DLV+WASP, GRINGO+CLASP, and GRINGO+WASP complete 40, 16, and 15 instances, respectively. I-DLV+CLASP is clearly the most performant ASP system of our baseline benchmark.

Our benchmark execution scripts, problem instance generator, and the instances described in Table 5 are available at `https://goo.gl/Xrd2Hq`.

5 Related Work

The complexity of the resource allocation problem in domains similar to BPM has been widely acknowledged in other research areas, such as Constraint Programming (CP) and Operations Research (OR). The survey conducted in (Lombardi and Milano, 2012) studied resource allocation and scheduling separately and jointly. Pure scheduling problems (which traditionally assume static task network and resource requirements) have been mostly addressed with CP techniques (Baptiste et al., 2001). Resource allocation, on the contrary, has been typically tackled as a MRCPSP with OR techniques (Brucker et al., 1999). For addressing large problem instances in real-world scenarios in which scheduling and resource allocation have to be jointly considered, hybrid CP/OR approaches were developed, including Logic Based Benders Decomposition (Hooker and Ottosson, 2003) and Self-adaptive Large Neighborhood Search (Laborie, 2009). In (Drexl et al., 2000), the authors extended the MRCPSP to consider further temporal aspects like travel times between tasks that have to be performed at two different locations by

the same resources. New concepts, such as partially renewable resources, were introduced and incorporated to the ProGen problem instance generator (Kolisch et al., 1995).

The work presented in this paper constitutes a realistic benchmark for declarative formalisms as our problem also involves the consideration of resource allocation and scheduling aspects. Specifically, the resource allocation problem in BPM is characterized by the essential ideas behind designing and managing business processes, and organizational models are crucial for extracting resource allocation requirements in BPM (Rosemann and vom Brocke, 2015). The extensive BPM literature with real use cases (Vom Brocke et al., 2010; Brocke and Rosemann, 2014; Dumas et al., 2018) is helpful for inferring those requirements. In our previous work, we addressed resource allocation in business processes with different problem configurations (i.e. Petri net semantics based encoding (Havur et al., 2015), extension of MRCPSP (Havur et al., 2016)). Nevertheless, Petri nets were encoded with ASP also in (Anwar et al., 2013), where the execution semantics of Petri nets is reflected for simulating biological pathways. However, in the work at hand, for improving the performance of our encoding we have eliminated these simulation-oriented aspects, which are redundant for resource allocation and scheduling purposes.

We believe ASP is a powerful formalism for addressing such a hard problem based on our previous work and on results found in the ASP Programming Competition, a biennial event aimed at providing challenging benchmark collections and evaluating the advancement of the state of the art in ASP solving (Gebser et al., 2017b). According to the categories defined based on the ASP features required for the encoding of the problems, resource allocation in business processes is an optimization problem (Gebser et al., 2017b). ASP has been successfully used for addressing optimization problems in other domains, as demonstrated in the 6th ASP Programming Competition, including, among others, Crossing Minimization (Gange et al., 2010), Valves Location (Gavanelli et al., 2015), MaxSAT (Li and Manyà, 2009), Steiner Tree (Erdem and Wong, 2004), System Synthesis (Biewer et al., 2015) and Video Streaming (Toni et al., 2014). The problem of resource allocation in business processes will be subjected to analysis within the 7th edition of the ASP Programming Competition (Gebser et al., 2017a).

6 Conclusions

We have modeled the resource allocation problem and provided an instance generator for benchmarking ASP systems. Our results show that this optimization problem is challenging for the current ASP systems which have received the highest performance scores in the 6th ASP Programming Competition held in 2015 (Calimeri et al., 2016).

Future research on optimizing the problem encoding for improving its computational efficiency might extend the applicability of our approach in solving the resource allocation in BPM in large-scale by using ASP. Apart from it, it is also on our agenda to extend our benchmark towards a more detailed comparison of many formalisms, and their systems might prove quite beneficial to both the LP and the BPM communities: the LP community would be provided with an application-oriented problem domain for assessing the advances in formalisms (the ease of encoding the resource allocation problem, i.e., the compactness, readability, modularity and maintainability of problem encoding) and the computational performance of their solvers; and the BPM community would be given a solid evaluation of different approaches and encouraged to try these resource allocation methods out-of-the-box, preferably via an interface to already existing BPMSs.

References

ALVIANO, M., CALIMERI, F., DODARO, C., FUSCÀ, D., LEONE, N., PERRI, S., RICCA, F., AND VELTRI, P. 2017. The ASP System DLV2. In *Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, M. Balduccini and T. Janhunen, Eds. Vol. 10377. Springer, 215–221.

ALVIANO, M., DODARO, C., LEONE, N., AND RICCA, F. 2015. Advances in WASP. In *Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, F. Calimeri, G. Ianni, and M. Truszczynski, Eds. Vol. 9345. Springer, 40–54.

ANWAR, S., BARAL, C., AND INOUE, K. 2013. Encoding Petri Nets in Answer Set Programming for Simulation Based Reasoning. *CoRR abs/1306.3542*, 1–17.

BAPTISTE, P., PAPE, C. L., AND NUIJTEN, W. 2001. *Constraint-Based Scheduling*. Kluwer Academic Publishers.

BIEWER, A., ANDRES, B., GLADIGAU, J., SCHAUB, T., AND HAUBELT, C. 2015. A Symbolic System Synthesis Approach for Hard Real-time Systems Based on Coordinated SMT-solving. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, W. Nebel and D. Atienza, Eds. ACM, 357–362.

BONET, B., HASLUM, P., HICKMOTT, S., AND THIÉBAUX, S. 2008. Directed unfolding of petri nets. In *Transactions on Petri Nets and Other Models of Concurrency I*. Springer, 172–198.

BREWKA, G., EITER, T., AND TRUSZCZYŃSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

BROCKE, J. V. AND ROSEMANN, M. 2014. *Handbook on Business Process Management 2: Strategic Alignment, Governance, People and Culture*. Springer.

BRUCKER, P., DREXL, A., MHRING, R., NEUMANN, K., AND PESCH, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research 112,* 1, 3–41.

BUCCAFURRI, F., LEONE, N., AND RULLO, P. 1997. Adding weak constraints to disjunctive datalog. In *APPIA-GULP-PRODE*. 557–568.

BURATTIN, A. 2015. Introduction to Business Processes, BPM, and BPM Systems. In *Process Mining Techniques in Business Environments*. Vol. 207. Springer, 11–21.

CALIMERI, F., FABER, W., GEBSER, M., IANNI, G., KAMINSKI, R., KRENNWALLNER, T., LEONE, N., RICCA, F., AND SCHAUB, T. 2013. ASP-Core-2: Input language format. Tech. rep., ASP Standardization Working Group.

CALIMERI, F., FUSCÀ, D., PERRI, S., AND ZANGARI, J. 2017. I-DLV: the new intelligent grounder of DLV. *Intelligenza Artificiale 11,* 1, 5–20.

CALIMERI, F., GEBSER, M., MARATEA, M., AND RICCA, F. 2016. Design and results of the fifth Answer Set Programming competition. *Artificial Intelligence 231*, 151–181.

CALIMERI, F., IANNI, G., AND RICCA, F. 2014. The third open Answer Set Programming competition. *Theory and Practice of Logic Programming 14,* 1, 117–135.

COLANTONIO, A., DI PIETRO, R., OCELLO, A., AND VERDE, N. V. 2009. A formal framework to elicit roles with business meaning in RBAC systems. In *ACM symposium on Access control models and technologies (SACMAT)*, B. Carminati and J. Joshi, Eds. ACM, 85–94.

COUVREUR, J.-M., POITRENAUD, D., AND WEIL, P. 2013. Branching processes of general petri nets. *Fundamenta Informaticae 122,* 1-2, 31–58.

DENECKER, M., VENNEKENS, J., BOND, S., GEBSER, M., AND TRUSZCZYŃSKI, M. 2009. The second Answer Set Programming competition. In *Int. Conf. on Logic Programming*

and Nonmonotonic Reasoning (LPNMR), E. Erdem, F. Lin, and T. Schaub, Eds. Vol. 5753. Springer, 637–654.

DREXL, A., NISSEN, R., PATTERSON, J. H., AND SALEWSKI, F. 2000. ProGen/x An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *European Journal of Operational Research 125,* 1, 59 – 72.

DUMAS, M., ROSA, M. L., MENDLING, J., AND REIJERS, H. A. 2018. *Fundamentals of Business Process Management (Second Edition)*. Springer.

ERDEM, E. AND WONG, M. D. F. 2004. Rectilinear Steiner Tree Construction Using Answer Set Programming. In *Int. Conf. on Logic Programming (ICLP)*, B. Demoen and V. Lifschitz, Eds. Vol. 3132. Springer, 386–399.

GANGE, G., STUCKEY, P. J., AND MARRIOTT, K. 2010. Optimal k-Level Planarization and Crossing Minimization. In *Int. Symposium on Graph Drawing (GD)*, U. Brandes and S. Cornelsen, Eds. Vol. 6502. Springer, 238–249.

GAREY, M. R., JOHNSON, D. S., AND SETHI, R. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research 1,* 2, 117–129.

GAVANELLI, M., NONATO, M., AND PEANO, A. 2015. An ASP approach for the valves positioning optimization in a water distribution system. *Journal of Logic and Computation 25,* 6, 1351–1369.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., ROMERO, J., AND SCHAUB, T. 2015. Progress in clasp series 3. In *Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, F. Calimeri, G. Ianni, and M. Truszczynski, Eds. Vol. 9345. Springer, 368–383.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., AND SCHAUB, T. 2012. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning 6,* 3, 1–238.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., AND SCHAUB, T. 2014. Clingo = ASP + Control: Extended Report. Tech. rep., University of Potsdam, Germany.

GEBSER, M., KAMINSKI, R., KÖNIG, A., AND SCHAUB, T. 2011. Advances in *gringo* series 3. In *Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, J. P. Delgrande and W. Faber, Eds. Vol. 6645. Springer, 345–351.

GEBSER, M., MARATEA, M., AND RICCA, F. 2017a. The Design of the Seventh Answer Set Programming Competition. In *Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, M. Balduccini and T. Janhunen, Eds. Vol. 10377. Springer, 3–9.

GEBSER, M., MARATEA, M., AND RICCA, F. 2017b. The Sixth Answer Set Programming Competition. *Journal of Artificial Intelligence Research 60*, 41–95.

HAVUR, G., CABANILLAS, C., MENDLING, J., AND POLLERES, A. 2015. Automated Resource Allocation in Business Processes with Answer Set Programming. In *BPM Workshops (BPI)*. Vol. 256. Springer, 191–203.

HAVUR, G., CABANILLAS, C., MENDLING, J., AND POLLERES, A. 2016. Resource Allocation with Dependencies in Business Process Management Systems. In *Int. Conf. on Business Process Management (BPM) - Forum*. Vol. 260. Springer, 3–19.

HOOKER, J. AND OTTOSSON, G. 2003. Logic-based Benders decomposition. *Mathematical Programming 96,* 1, 33–60.

HORLING, B. AND LESSER, V. 2004. A Survey of Multi-agent Organizational Paradigms. *Knowledge Engineering Review 19,* 4, 281–316.

KOLISCH, R., SPRECHER, A., AND DREXL, A. 1995. Characterization and Generation of a General Class of Resource-constrained Project Scheduling Problems. *Management Science 41,* 10, 1693–1703.

LABORIE, P. 2009. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, W.-J. van Hoeve and J. N. Hooker, Eds. Vol. 5547. Springer, 148–162.

LI, C. M. AND MANYÀ, F. 2009. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, 613–631.

LOHMANN, N., VERBEEK, E., AND DIJKMAN, R. 2009. Petri Net Transformations for Business Processes - A Survey. *Trans. on Petri Nets and Other Models of Concurrency II 2*, 46–63.

LOMBARDI, M. AND MILANO, M. 2012. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints 17*, 51–85.

OMG. 2011. BPMN 2.0. Recommendation, OMG.

POPOVA-ZEUGMANN, L. 2013. Time Petri Nets. In *Time and Petri Nets*. Springer, 139–140.

ROGGE-SOLTI, A. 2014. Block-structured stochastic Petri net generator (ProM plug-in). `http://www.promtools.org/`. Accessed: 2018-01-01.

ROSEMANN, M. AND VOM BROCKE, J. 2015. The six core elements of business process management. In *Handbook on Business Process Management 1*. Springer, 105–122.

RUSSELL, N., VAN DER AALST, W. M., TER HOFSTEDE, A. H., AND EDMOND, D. 2005. Workflow resource patterns: Identification, representation and tool support. In *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, O. Pastor and J. F. e Cunha, Eds. Vol. 3520. Springer, 216–232.

SCHÖNIG, S., CABANILLAS, C., JABLONSKI, S., AND MENDLING, J. 2016. A Framework for Efficiently Mining the Organisational Perspective of Business Processes. *Decision Support Systems 89*, 87–97.

SPRECHER, A., HARTMANN, S., AND DREXL, A. 1994. Project scheduling with discrete time-resource and resource-resource tradeoffs. Tech. rep., Instituten für Betriebswirtschaftslehre der Universität Kiel, Germany.

TONI, L., APARICIO-PARDO, R., SIMON, G., BLANC, A., AND FROSSARD, P. 2014. Optimal Set of Video Representations in Adaptive Streaming. In *ACM Multimedia Systems Conference (MMSys)*, R. Zimmermann, Ed. ACM, 271–282.

VAN DER AALST, W. M. 1996. Structural characterizations of sound workflow nets. *Computing Science Reports 96,* 23, 18–22.

VAN DER AALST, W. M. P. 2016. *Process Mining - Data Science in Action (Second Edition)*. Springer.

VOM BROCKE, J., ROSEMANN, M., ET AL. 2010. *Handbook on business process management*. Springer.