Expert Systems With Applications 205 (2022) 117599

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Benchmarking Answer Set Programming systems for resource allocation in business processes

Giray Havur [a,b], Cristina Cabanillas [c], Axel Polleres [a,d,*]

[a] Vienna University of Economics and Business, Vienna, Austria
[b] Siemens AG Österreich, Technology, Vienna, Austria
[c] SCORE Lab & I3US Institute, Universidad de Sevilla, Seville, Spain
[d] Complexity Science Hub, Vienna, Austria

ARTICLE INFO

Keywords:
Resource allocation
Business process management
Answer set programming
Benchmark

ABSTRACT

Declarative logic programming formalisms are well-suited to model various optimization and configuration problems. In particular, Answer Set Programming (ASP) systems have gained popularity, for example, to deal with scheduling problems present in several domains. The main goal of this paper is to devise a benchmark for ASP systems to assess their performance when dealing with complex and realistic resource allocation with objective optimization. To this end, we provide (i) a declarative and compact encoding of the resource allocation problem in ASP (compliant with the ASP Core-2 standard), (ii) a configurable ASP systems benchmark named BRANCH that is equipped with resource allocation instance generators that produce problem instances of different sizes with adjustable parameters (e.g., in terms of process complexity, organizational and temporal constraints), and (iii) an evaluation of four state-of-the-art ASP systems using BRANCH. This solid application-oriented benchmark serves the ASP community with a tool that leads to potential optimizations and improvements in encodings and further drives the development of ASP solvers. On the other hand, resource allocation is an important problem that still lacks adequate automated tool support in the context of Business Process Management (BPM). The ASP problem encoding, ready-to-use ASP systems and problem instance generators benefit the BPM community to tackle the problem at scale and mitigate the lack of openly available problem instance data.

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, Rosa, Mendling, & Reijers, 2018). As an integral part of business processes, resources have to be considered throughout all the stages of the BPM life cycle, which iterates from process discovery and modeling to process execution and monitoring, targeting continuous process improvement. At run time, a process execution engine creates business process instances and allocates specific resources to the tasks to be completed according to predefined criteria. Such criteria include, for example, constraints that comprise the characteristics of resources needed for each process task. Regarding human resources, these characteristics are usually reflected in organizational models that contain all the relevant data about the resources, such as their roles, skills and any other valuable information. For instance, Role-Based Access Control (RBAC) (Colantonio, Pietro, Ocello, & Verde, 2009) is very often used for assigning resources to process tasks based on the organizational roles they are associated with. As a result, during process execution, at the due time for each activity only the required resource will be picked up from the set of candidates (i.e., among suitable resources according to the resource assignment constraints) and allocated to the task. Such a selection and scheduling of resource assignments is addressed in Resource Allocation in Business Processes (RABP).

An RABP problem comprises several components: a process model that describes the control flow of activities (e.g., precedence and concurrency relations), an organizational model that characterizes resources and their roles to enable activity executions (Russell, van der Aalst, ter Hofstede, & Edmond, 2005), and a temporal model that designates activity duration estimations. Moreover, process-oriented

* Corresponding author.
E-mail addresses: giray.havur@wu.ac.at (G. Havur), cristinacabanillas@us.es (C. Cabanillas), axel.polleres@wu.ac.at (A. Polleres).

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

organizations are also concerned with carrying out RABP under optimization objectives so that one or many process performance measures (e.g., execution time and cost) are optimized (Dumas et al., 2018), which adds to the complexity of RABP. One way of approaching this problem is by encoding it using a declarative programming formalism. Answer Set Programming (ASP), a declarative logic programming dialect particularly suitable to model combinatorial search problems, appears to be 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, Kaminski, Kaufmann, & Schaub, 2012). ASP programs consist of clauses that look similar to Prolog rules (Colmerauer & Roussel, 1993) but the underlying computational mechanisms, which are based on the stable model semantics (Gelfond & Lifschitz, 1988), are different (Lifschitz, 2008): rather than via resolution-based proofs, an ASP program is first *grounded* to a finite set of clauses where each rule with variables is instantiated as equivalent propositional rules without variables. Afterward, a *solver* searches for answer sets (i.e., solutions) of the ground program. This is typically done by relying on advanced conflict-driven, heuristic search procedures developed in the area of propositional satisfiability checking (SAT) (Drescher, Gebser, Kaufmann, & Schaub, 2010). Moreover, apart from rules and hard constraints, in the absence of complete information, *default behaviors* related to an allocated resource (e.g., assumed/default duration of a generic activity, potentially overridden by a known different duration for a specific resource/role) can be elegantly represented in ASP by so-called negation-as-failure (Gelfond & Lifschitz, 1988; Lifschitz, 2008). Due to its flexible modeling constructs, in prior research efforts (Havur, Cabanillas, Mendling, & Polleres, 2015, 2016) we could demonstrate the suitability of ASP as a flexible tool to encode RABP with makespan[1] minimization, including relatively complex constraints. This allowed us to identify the RABP problem as a challenging task for state-of-the-art ASP systems (i.e., combinations of an ASP grounder and an ASP solver); however, so far, a comprehensive set of realistic benchmark instances is missing to stress-test the performance of ASP systems in practice on RABP.

The ASP community has collected benchmarks to test ASP systems by organizing regular competitions. These competitions have both been successful in driving research on more efficient grounders and solvers and optimizing and comparing problem encodings to evaluate the different performance of otherwise equivalent encodings in different ASP systems. Most closely related to our problem, earlier such ASP competitions (Denecker, Vennekens, Bond, Gebser, & Truszczynski, 2009; Gebser, Maratea, & Ricca, 2020) have led to the development of two scheduling-related benchmark entries called disjunctive scheduling (Denecker et al., 2009) and incremental scheduling (Gebser et al., 2020). Unlike these two problems that were addressed in two earlier ASP competitions, the RABP problem has a makespan optimization criteria and a more elaborate and expressive input/output setting that is tailored for real-world applicability, for which a benchmark to compare the performance of existing ASP solvers and grounders is still missing. In the case of RABP, this is also due to the lack of openly available real-world data describing various scenarios in which resource allocation is required (i.e., the lack of ready-to-use datasets), as companies typically consider information about their resources or business processes sensitive.

In this paper, we aim at narrowing this gap by extending former contributions on ASP-based RABP towards a common challenge benchmark for ASP solvers, driven by and parameterizable to mimic realistic scenarios from BPM. In particular, our contribution is three-fold:

- We define a formalization of the RABP problem and provide a *baseline problem encoding in ASP*.
- We develop a ready-to-use, configurable *benchmark* named BRANCH that generates RABP instances with respect to given parameters to adapt the instance sizes, solves the generated RABP instances using the configured ASP systems by the users, and reports on the performance of the ASP systems.
- Lastly, by using BRANCH, we present a detailed *evaluation* that compares four ASP systems comprising combinations of state-of-the-art ASP grounders GRINGO (Gebser, Kaminski, König, & Schaub, 2011) and I-DLV (Calimeri, Fuscà, Perri, & Zangari, 2017), and solvers CLASP (Gebser, Kaminski, Kaufmann, Romero, & Schaub, 2015) and WASP (Alviano, Dodaro, Leone, & Ricca, 2015).

Interestingly, the results of our evaluation demonstrate that real-world instance sizes pose a considerable challenge on state-of-the-art ASP systems, which opens further opportunities for specific performance improvements in these systems. While this might be a valuable insight for the ASP research community, the BPM community will also benefit from a new, declarative encoding of the RABP problem, available systems to solve it, and a generator to produce instance datasets for testing purposes.

The remainder of this paper is structured as follows. Section 2 formally defines the RABP problem, providing the necessary background. Based on that, Section 3 presents our baseline problem encoding, preceded by a detailed background on ASP. Section 4 describes the functionalities of BRANCH. Section 5 evaluates the performance of different ASP systems using our baseline ASP encoding with BRANCH. Section 6 highlights the limitations of BRANCH. Finally, Section 7 concludes the paper, summarizing insights and giving pointers for future work.

2. Formalization of the problem

In this section, we first describe the elements involved in RABP, such as business process models, organizational models, and the modeling of temporal aspects. Afterward, we formalize the RABP problem.

2.1. Business process models

A business process is a finite set of structured activities where a specific sequence of activities serves a particular business goal. The execution of each activity in the process generally requires one human resource and produces an output that is of benefit for a customer, such as a service or a product (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. As for formal descriptions of business processes, in practice, Business Process Model and Notation (BPMN) (OMG, 2014) diagrams are widely used to describe business processes due to their visual understandability.

Book Publishing (BPub) Example 1. Fig. 1 depicts the model of a process for publishing a book from the point of view of a publishing house, represented as a BPMN diagram. In this process, when the publishing entity receives a new textbook manuscript from an author, it must be proofread and revised. The improved manuscript is then sent back to the author (press release) for double-checking.

High-level, visual process modeling notations like BPMN can be automatically mapped to Petri nets (Dijkman, Dumas, & Ouyang, 2008; Peterson, 1981), which have the advantage of an exact mathematical definition of their execution semantics and their well-developed theory for process analysis.

Definition 1 (*Petri Net*)**.** A *Petri net* is a 3-tuple $PN = (P, T, F)$ where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a set of *places*, represented graphically as circles;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a set of *transitions*, represented graphically as rectangles; and
- $F \subseteq (P \times T) \bigcup (T \times P)$ is a set of *arcs* (flow relations), represented as arrows and describing a bipartite graph.

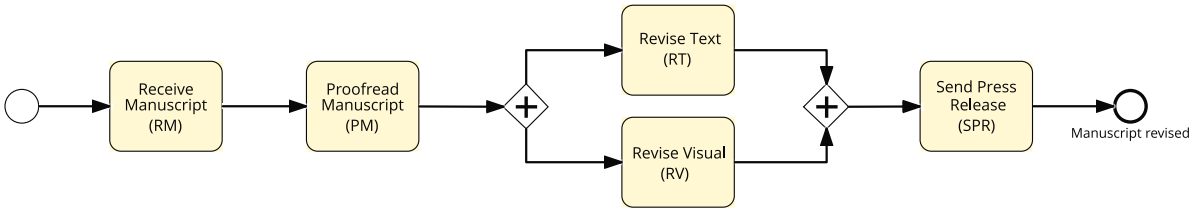


Fig. 1. BPMN model of the book publishing process.

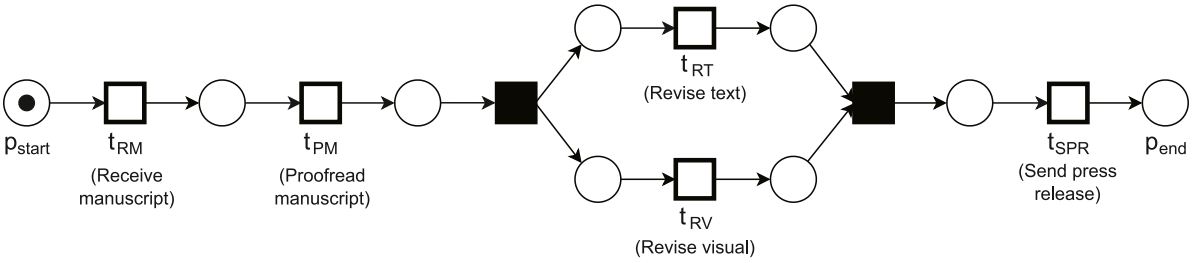


Fig. 2. Petri net model of the book publishing process.

Table 1
Behavioral relation matrix of the activities in Fig. 1.

	RM	PM	RT	RV	SPR
RM		$\succ, \rightarrow$	$\succ, \rightarrow$	$\succ, \rightarrow$	$\succ, \rightarrow$
PM			$\succ, \rightarrow$	$\succ, \rightarrow$	$\succ, \rightarrow$
RT				$\succ, \|\|$	$\succ, \rightarrow$
RV			$\succ, \|\|$		$\succ, \rightarrow$
SPR					

The *input places* and the *output places* of each transition $t \in T$ are defined as ${}^\bullet t = \{p \in P \mid (p,t) \in F\}$ 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 defined as ${}^\bullet p = \{t \in T \mid (t,p) \in F\}$ and $p^\bullet = \{t \in T \mid (p,t) \in F\}$, respectively. $\mu : P \rightarrow \mathbb{N}_0$ is a marking of PN representing the initial distribution of tokens and $\mu(p)$ maps a place p to its number of tokens. Pictorially, tokens are represented as black dots in places (e.g., $\mu(p_{start}) = 1$ in Fig. 2). A transition t is *enabled* in μ, denoted by $(PN, \mu)[t\rangle$, when there is at least one token in each input place of t where $p \in {}^\bullet t$. An enabled transition can therefore *fire*. The firing of a transition t changes the current marking μ to μ' by removing one token from each $p \in {}^\bullet t$ and adding one token to each $p \in t^\bullet$, denoted by $(PN, \mu)[t\rangle(PN, \mu')$. A firing sequence of transitions $\sigma = t_1, t_2, \ldots, t_n$ changes the *initial marking* of the Petri net μ_0 at each firing. If there exists a firing sequence σ leading to μ', then μ' is *reachable* from μ_0, denoted by $\mu' \in [PN, \mu_0\rangle$. A *Petri net system* is a pair (PN, μ_0).

BPub Example 2. Fig. 2 illustrates the Petri net representation of the book publishing process described above, corresponding to the BPMN diagram in Fig. 1. Activities $A \subseteq T$ are transitions that are graphically represented by empty squares.

Among the *behavioral relations* that are described in Weidlich, Mendling, and Weske (2011), between two different transitions t_x and t_y of a Petri net system (PN, μ_0):

- $t_x \succ t_y$ reads as t_x is in the *weak order relation* with t_y if there exists a firing sequence $\sigma = t_1, t_2, \ldots, t_n$ such that $(PN, \mu_0)[\sigma\rangle$, $x \in \{1, \ldots, n-1\}$ and $x < y \leq n$;
- $t_x \rightarrow t_y$ reads as t_x is in the *precedence relation* with (i.e., *precedes*) t_y
- $t_x \parallel t_y$ reads as t_x is in the *concurrency relation* with t_y if there is a reachable marking $\mu' \in [PN, \mu_0\rangle$ that enables both transitions concurrently.

BPub Example 3. The behavioral relations between the activities in Fig. 1 are derived from the equivalent Petri net in Fig. 2. These relations are summarized in Table 1. For example, $PM \succ RT$, $PM \rightarrow RT$, and $RT \parallel RV$.

2.2. Organizational model

Different types of organizational structures give rise to different organizational models (Horling & Lesser, 2004). In most cases, the employees of the organization (i.e., its human resources) have one or more organizational roles according to their skills and characteristics, which allow them to perform certain tasks and activities. One of the most widely known models for capturing such organizational structures based on roles is the Role-Based Access Control (RBAC) model (Colantonio et al., 2009). It describes resources, roles, and *who* can execute *which* activity based on the roles.

Definition 2 (*RBAC Model*). An *RBAC Model* is a 6-tuple $O = (A, R, L, S_{AL}, S_{RL}, S_{LL})$, where:

- A is a set of *activities*,
- R is a set of *resources*,
- L is a set of *roles*,
- $S_{AL} \subseteq 2^{(A \times L)}$ is a set of activity-to-role assignments specifying which activity can be executed by which role(s) (i.e., the resources associated with these roles), commonly known as *resource assignment* in BPM (Cabanillas, Resinas, del-Río-Ortega, & Cortés, 2015),
- $S_{RL} \subseteq 2^{(R \times L)}$ is the corresponding set of resource-to-role assignment tuples identifying the roles per resource,
- $S_{LL} \subseteq 2^{(L \times L)}$ is a set of role-to-role assignments that form a hierarchical (sub-role) structure. The symbol $\geq$ indicates the ordering operator. If $l_1 \geq l_2$, then l_1 is referred to as the *senior* of l_2. Conversely, l_2 is the *junior* of l_1 with the intuitive meaning that each senior role may also execute all activities that the junior role is allowed to execute.

BPub Example 4. Let us assume that the book publishing process (cf. Figs. 1 and 2) is executed within an organization composed of six resources $R = \{Amy, Glen, Drew, Emily, Oliver, Evan\}$ that are assigned to four distinct roles $L = \{Publisher, Copy\ Editor, Graphic\ Artist, Administrative\ Assistant\}$. Table 2 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,

Table 2
RBAC model of the publishing entity.

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

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 all the activities of the resources with the role *Copy Editor* but not the other way around (specifically, *Publisher* $\geq$ *Copy Editor*).

2.3. Temporal model

In a typical real-world BPM scenario, temporal constraints like durations of activities and the tentative deadline for the completion of process instances (i.e., makespan upper bound) are estimated by process managers.

Definition 3 (*Default Duration Function*)**.** The default duration function $\Delta : A \to \mathbb{N}_0$ defines the activity duration in a use-case specific time unit (TU) (e.g., in minutes, hours or days) for each activity $a \in A$.

We may assume the temporal model is typically created by process managers.[2] However, to increase the robustness of resource allocation, *resource-activity*, and *role-activity* specific durations can be estimated from an event log using process mining techniques (van der Aalst, 2016). Process execution data from past process instances is typically stored in event logs or audit trails by BPM Systems (BPMS) and other information systems used in the organization (van der Aalst & van Dongen, 2013). An event log usually stores for each event related to an activity the type of event (e.g., start or end of an activity execution), resource-related properties (at least *who* executed an activity[3]), and temporal information (*when* the event took place). All the events related to a process instance constitute a *trace*. Each event and each trace in an event log are typically identified by a unique event id and trace id, respectively.

Definition 4 (*Resource-Activity Duration Partial Function*)**.** The resource-activity duration partial function $\delta_r : (\mathcal{L} \times R \times A) \nrightarrow \mathbb{N}_0$ estimates the mean duration for executing an activity a by a resource r from an event log $\mathcal{L}$.

Definition 5 (*Role-Activity Duration Partial Function*)**.** The role-activity duration partial function $\delta_l : (\mathcal{L} \times L \times A) \nrightarrow \mathbb{N}_0$ estimates the mean duration for executing an activity a by a role l from an event log $\mathcal{L}$ where $(r, l) \in S_{RL}$ and $(a, l) \in S_{AL}$.

[2] Process managers design processes and implement improvements in them as needed.

[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).

BPub Example 5. Table 3 shows an excerpt $\mathcal{L}'_{\text{BPub}}$ of the event log $\mathcal{L}_{\text{BPub}}$ from the execution of the book publishing process depicted in Fig. 2. For example, the second row (with the event id e57) is logged at the start of the execution of the activity *Proofread Manuscript* by *Glen* on 2021-10-15 at 09:16. An example of role-activity duration estimation from an event log is as follows: given the partial event log $\mathcal{L}'_{\text{BPub}}$ in Table 3, $\pi(\mathcal{L}'_{\text{BPub}}, Copy\ Editor, Proofread\ Manuscript)$ is estimated by calculating the mean of the two time intervals e57 to e64 and e59 to e60 (i.e., the copy editors *Glen* and *Drew*'s execution intervals of *Proofread Manuscript*). The first time interval is from 09:16 to 13:26 (250 min), and the second one is from 09:35 to 12:45 (190 min). Therefore, the mean of the two durations is estimated to be 220 min. Resource-activity duration estimations are calculated in a similar fashion. Associated with our running example, Table 4 shows the temporal model including the resource-activity and role-activity durations that are assumed to be extracted from the *complete* event log $\mathcal{L}_{\text{BPub}}$, and the default activity durations that are assumed to be estimated by the process manager while designing the process.

The selection of the duration to be considered in the allocation for a resource r's execution of an activity a is defined by the resource-activity duration preference function.

Definition 6 (*Resource-Activity Duration Preference Function*)**.** The resource-activity duration preference function $\pi : (\mathcal{L} \times R \times A) \to \mathbb{N}_0$ handles the preference among the resource-activity duration $\delta_r(\mathcal{L}, r, a)$, role-activity duration $\delta_l(\mathcal{L}, l, a)$, and default duration of an activity $\Delta(a)$.

$$\pi(\mathcal{L}, r, a) = \begin{cases} \delta_r(\mathcal{L}, r, a) & \text{if } \delta_r(\mathcal{L}, r, a) \text{ is defined,} \\ \delta_l(\mathcal{L}, l, a) & \text{if } \delta_r(\mathcal{L}, r, a) \text{ is not defined and} \\ & (r, l) \in S_{RL}, \\ \Delta(a) & \text{otherwise.} \end{cases}$$

$\pi(\mathcal{L}, r, a)$ returns the resource-activity duration if $\delta_r(\mathcal{L}, r, a)$ has a value for the given parameters $\mathcal{L}$, r, and a. Otherwise, it returns the role-activity duration provided that $\delta_l(\mathcal{L}, l, a)$ has a value for the given parameters $\mathcal{L}$, l, and a where there is an activity-to-role assignment (r, l) in the RBAC model. When neither of the functions can be resolved, $\pi(\mathcal{L}, r, a)$ returns the default activity duration $\Delta(a)$, i.e., the estimated duration by the process manager while designing the process.

BPub Example 6. $\pi(\mathcal{L}_{\text{BPub}}, Emily, PM)$ returns 208 time units since Emily has never executed *Proofread Manuscript* herself in the past (i.e., there is no resource-activity duration for (*Emily*, *Proofread Manuscript*) in Table 4), but there is a role-activity duration for (*Copy Editor*, *Proofread Manuscript*) where (*Emily*, *Copy Editor*) $\in S_{RL}$.

2.4. Resource Allocation in Business Processes (RABP)

RABP aims at finding a feasible allocation consisting of a set of quadruples $I \subseteq 2^{(R \times A \times U \times U)}$ such that $(r_i, a_i, s_i, c_i) \in I$ where each activity $a_i \in A$ is assigned a resource $r_i \in R$, a start time $s_i \in U$, and a completion time $c_i = s_i + \pi(\mathcal{L}, r, a)$. It is assumed that each activity, once started, is planned to be completed without interruptions in the schedule (i.e. activities are non-preemptive). The following constraints (c.1-c.4) hold for RABP:

(c.1) Only one resource is allocated to each activity.
$\forall a_i \in A : |\{(r_i, a_i, s_i, c_i) \in I\}| = 1$

(c.2) Each activity has only one start time.
$\forall i_1, i_2 \in I : a_{i_1} = a_{i_2} \Rightarrow s_{i_1} = s_{i_2}$

(c.3) The start time of any activity is greater than or equal to the completion time of its preceding activities.
$\forall i_1, i_2 \in I : a_{i_1} \rightarrow a_{i_2} \Rightarrow s_{i_2} \geq c_{i_1}$

Table 3
The excerpt $\mathcal{L}'_{\text{BPub}}$ of the event log $\mathcal{L}_{\text{BPub}}$ from the execution of the book publishing process in Fig. 2 executed by resources defined in the RBAC model in Table 2.

event id	trace id	event type	activity	resource	time stamp
...					
e56	7	start	Receive Manuscript (RM)	Amy	2021-10-15T**08:25**:27
e57	6	start	Proofread Manuscript (PM)	Glen	2021-10-15T**09:16**:54
e58	7	end	Receive Manuscript (RM)	Amy	2021-10-15T**09:21**:20
e59	7	start	Proofread Manuscript (PM)	Drew	2021-10-15T**09:35**:13
e60	7	end	Proofread Manuscript (PM)	Drew	2021-10-15T**12:45**:35
e61	7	start	Revise Visual (RV)	Oliver	2021-10-15T**12:55**:01
e62	7	start	Revise Text (RT)	Drew	2021-10-15T**13:07**:08
e63	8	start	Receive Manuscript (RM)	Amy	2021-10-15T**13:15**:42
e64	6	end	Proofread Manuscript (PM)	Glen	2021-10-15T**13:26**:23
e65	6	start	Revise Text (RT)	Glen	2021-10-15T**14:47**:07
...					

Table 4
Default activity durations, resource-activity durations derived from the event log $\mathcal{L}_{\text{BPub}}$ and role-activity durations derived from the event log $\mathcal{L}_{\text{BPub}}$.

default activity durations	
Receive Manuscript	20
Proofread Manuscript	180
Revise Text	240
Revise Visual	240
Send Press Release	30
resource-activity durations	
(Amy, Receive Manuscript)	45
(Drew, Proofread Manuscript)	247
(Glen, Proofread Manuscript)	182
(Drew, Revise Text)	186
(Glen, Revise Text)	150
(Oliver, Revise Visual)	221
(Evan, Send Press Release)	55
role-activity durations	
(Publisher, Receive Manuscript)	45
(Copy Editor, Proofread Manuscript)	208
(Copy Editor, Revise Text)	171
(Graphic Artist, Revise Visual)	221
(Administrative Asst., Send Press Release)	55

(c.4) Same resource must not be allocated to any concurrent pair of activities that have overlapping execution periods.

$$\forall i_1, i_2 \in I : (a_{i_1} \parallel a_{i_2} \wedge s_{i_2} \leq s_{i_1} < c_{i_2}) \Rightarrow r_{i_1} \neq r_{i_2}$$
$$\forall i_1, i_2 \in I : (a_{i_1} \parallel a_{i_2} \wedge s_{i_2} < c_{i_1} \leq c_{i_2}) \Rightarrow r_{i_1} \neq r_{i_2}$$
$$\forall i_1, i_2 \in I : (a_{i_1} \parallel a_{i_2} \wedge s_{i_2} > s_{i_1} \wedge c_{i_2} < c_{i_1}) \Rightarrow r_{i_1} \neq r_{i_2}$$

3. Baseline ASP encoding of the problem

Before we delve into the details of the encoding of RABP in ASP (Section 3.3), let us introduce fundamentals of ASP (Section 3.1) and some theoretical motivation of why RBAP is particularly suitable to be encoded and solved with ASP (Section 3.2).

3.1. Fundamentals of ASP

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

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

where $n \geq m \geq 0$ and each a_i is a function-free first-order atom. The symbol ":-" is read as *if*: The left-hand side (e.g., a_0) is called the *head* of the rule and the right-hand side (e.g., $a_1, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n$) is called the *body* of the rule. Semantically, *if* the *body* holds then the *head* is derived. When the *body* is empty, the symbol ":-" is dropped and the rule is called a *fact* (e.g., a_0.). "*not*" is called *negation as failure*: *not* a_i is derived from failure to derive a_i. In rule (1), if $a_1, \ldots, a_m$ are true and none of $\mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n$ can be proven to be true then a_0 must be true. When the *head* is empty in a rule, we call it a *constraint*. Constraints rule out models satisfying their body atoms, which eliminates unwanted solution candidates.

Whenever a_i is a first-order predicate with variables within a rule r of the form (1), this rule is considered a shortcut for its "grounding" $g(r)$ (i.e., the set of its instantiations obtained by replacing the variables with all possible constants occurring in Π). Likewise, we denote by $g(\Pi)$ the set of rules obtained from grounding all rules in Π.

As a syntactic extension, the ASP Core-2 standard (Calimeri et al., 2020) allows set-like *choice expressions* of the form

$$x \leq \{a_1, \ldots, a_m\} \leq y \ :- \ a_n, \ldots, a_k, \mathit{not}\ a_{k+1}, \ldots, \mathit{not}\ a_l.$$

that is, if the *body* holds then an arbitrary subset of $\{a_1, \ldots, a_m\}$ of minimum size of x and maximum size of y is derived.

Example 3.1. The following ASP program describes a simplified resource allocation setting for exemplifying the concepts in ASP. In this resource allocation setting, the activities have no behavioral relation (i.e., precedence and concurrency relations) in contrast to RABP.

```
resource(r1). resource(r2). resource(r3).                    1
activity(a1;a2).                                             2

1<={allocation(R,A) : resource(R)}<=1                        3
:- activity(A).                                              4

:- allocation(R,A1), allocation(R,A2), A1!=A2.               5

:- allocation(r1,a2).                                        6
:- allocation(r2,a1).                                        7
```

Lines (1,2) are facts stating that `r1`, `r2` and `r3` are resources; and `a1` and `a2` are activities – "`activity(a1;a2).`" is a syntactic shortcut for "`activity(a1). activity(a2).`". Line (3) is a choice expression that generates the `allocation` predicates. This line can be read as follows: for each activity `a` ∈ {`a1`,`a2`}, derive minimum 1 and maximum 1 (i.e., only one) `allocation(r',a')` where `a'` is equal to `a` and `R` is selected from the domain of the predicate `resource` (i.e., `r'` ∈ {`r1`,`r2`,`r3`}). Lines (4–6) are constraints. Line (4) omits the answers in which a resource is allocated to more than one activity. Line (5) prohibits the resource `r1` to be allocated to the activity `a2`, and line (6) prohibits the resource `r2` to be allocated to the activity `a1`. The three different answers (i.e., solutions) of this program are as follows:

```
A 1: allocation(r1,a1) allocation(r2,a2)
A 2: allocation(r3,a1) allocation(r2,a2)
A 3: allocation(r1,a1) allocation(r3,a2)
```

Aggregates (Gebser, Kaminski, Ostrowski, Schaub, & Thiele, 2009) are arithmetic operations over a set of elements and they occur in aggregate atoms in rule bodies that have the form $\#aggr\{w_1 : a_1, \ldots, w_n : a_n\} \oplus v$, where w_i is the weight assigned to a_i; the operation $aggr \in$ {"`count`" ,"`sum`","`min`", "`max`"}; the comparison operator $\oplus \in$ {"$<$","$\leq$","$=$","$\neq$", "$>$","$\geq$"}; and v is a bound value. For instance,

the aggregate $\#\mathtt{sum}\{w_1 : a_1, \ldots, w_n : a_n\} \oplus v$ is interpreted as *true* if $\sum_{i=1}^{n} w_i \oplus v$ holds, and $false$ otherwise. In addition, an aggregate on the right-hand side of the assignment operator "=" may also be used for assigning the result obtained from the aggregate's evaluated value to a variable.

Example 3.2. Following Example 3.1, we add the following lines in our program for taking into account the execution times required for each resource to execute an activity.

```
execTime(r1,a1,10). execTime(r1,a2,10).                    7
execTime(r2,a1,20). execTime(r2,a2,20).                    8
execTime(r3,a1,30). execTime(r3,a2,30).                    9

totalTime(T) :-
    T=#sum{E, A : allocation(R,A), execTime(R,A,E)}.       10
```

Lines (7–9) represent the execution times of activities by specific resources. For example, `execTime(r1,a1,10)` means that the resource `r1` executes the activity `a1` in 10 min. Line (10) sums the total execution time of all activities with respect to the allocated resources. The program described in lines (1–10) returns the following three answers:

```
A 1: allocation(r1,a1) allocation(r2,a2) totalTime(30)
A 2: allocation(r1,a1) allocation(r3,a2) totalTime(40)
A 3: allocation(r3,a1) allocation(r2,a2) totalTime(50)
```

Weak constraints (Leone et al., 2006) allow us to formalize optimization problems in an easy and natural way. In such problems we use weak constraints to indicate preferences between possible answer sets:

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

where $t_1, \ldots, t_n$ are terms (e.g., atoms), and w is a weight. The sum of weights w over all occurrences of weighted atoms that are satisfied by a stable model are therefore minimized.

Example 3.3. We add a weak constraint to our example as follows:

```
:~ totalTime(T). [T, T]                                    11
```

Line (11) asks the solver to minimize the variable `T` of `totalTime` in the answer. The program described in lines (1–11) returns the following optimum answer:

```
OPT: allocation(r1,a1) allocation(r2,a2) totalTime(30)
```

Therefore, the optimum answer with the minimum value of the `totalTime` variable `T` is returned as an answer (cf. Example 3.2).

Sets of rules are evaluated in ASP under the *stable-model semantics* (Gelfond & Lifschitz, 1988), which allows several models (so-called *answer sets*) . The ASP systems typically first compute a $g(\Pi)$ via a *grounder*, and then use a Davis–Putnam–Logemann–Loveland- (DPLL)-like branch-and-bound algorithm to find answer sets for this ground program via a *solver*. There are various *grounders* and *solvers* for solving problems encoded in ASP (Gebser, Maratea, & Ricca, 2017). They vary in terms of the meta-heuristics and extensions incorporated in their implementations. We refer to Brewka et al. (2011) and references therein for details.

3.2. Theoretical motivation: Why to solve RABP with ASP?

In order to motivate why the RABP problem as introduced in this paper is amenable to be solved with ASP, let us argue by comparing RABP to related problems from the literature.

A couple of scheduling-related benchmarks have been presented and executed in the earlier ASP competitions (Denecker et al., 2009; Gebser et al., 2020). The first entry is called disjunctive scheduling (Denecker et al., 2009) and assumes generalized activity precedence relations (i.e., any activity can be preceded by any other activities without further limitations, such as those imposed in RABP due to the need of having well-formed business process structures), uniform activity durations (i.e., all activities have the same execution time), and an overall deadline to be met without additional resource-related constraints. The other related entry is called incremental scheduling (Gebser et al., 2020) and addresses a problem with one renewable resource[4] set, fixed activity durations (i.e., each activity has only one possible execution time) and deadlines imposed on activities. Unlike the problems involved in these two ASP competition entries, the RABP problem has different numbers of resource sets (i.e., roles), resource- and resource-set-specific time requirements and – as the overall optimization criterion – minimization of the total makespan.

Allocation of resources with starting times to the activities (i.e., RABP) is a far-from-trivial task with strong implications on the quality of the final allocation, and it is incredibly challenging from a computational perspective (Lombardi & Milano, 2012). As for complexity, RABP with makespan optimization is NP-Hard: the Job Shop Scheduling Problem (JSSP), known to be NP-Complete for its variants with two or more machines (Garey, Johnson, & Sethi, 1976), is polynomial-time reducible to a RABP problem. A JSSP instance can simply be translated into a RABP problem instance where (i) the tasks of different jobs that require the same machine would be represented as activities in concurrency relation, (ii) the order of tasks in jobs maps to precedence relations of activities, (iii) machines map to roles with only one resource, and (iv) task durations for specific machines are represented as default activity durations. Since we provide an encoding of RABP in ASP with weak constraints, which captures the class of Δ_2^P that is typically used for computing optimal solutions among such NP-complete problems with a given upper bound u to be minimized (Buccafurri, Leone, & Rullo, 1997), we also show that RABP is in Δ_2^P.

Moreover, ASP provides a simple and elegant representation to define and handle the preferences/priorities of resource-bound activity durations and also default activity durations in absence of complete information. For these reasons, therefore, the RABP problem seems to be an ideal candidate for benchmarking ASP systems supporting weak constraints.

3.3. ASP encoding of RABP

Our encoding is shown in Fig. 3. It encapsulates the formalization in Section 2.4 in a straightforward, declarative manner (i.e., no encoding "tricks" that optimize for a specific ASP grounder or solver have been applied).

Input Format. The input is divided into three groups of predicates as follows.

Behavioral relations of activities in a business process $\mathcal{P}$ as described in Section 2.1:

- `activity`(a): a is an activity in $\mathcal{P}$;
- `prec`(a_1, a_2): the activity a_1 precedes the activity a_2;
- `conc`(a_1, a_2): the activity a_1 is concurrency with the activity a_2.

An RBAC organizational model $O = (A, R, L, S_{AL}, S_{RL}, S_{LL})$ as described in Section 2.2:

- `alAC`(a, l): the 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): the resources with role l_1 can execute the same activities as the resources with role l_2 (i.e., $(l_1, l_2) \in S_{LL}$).

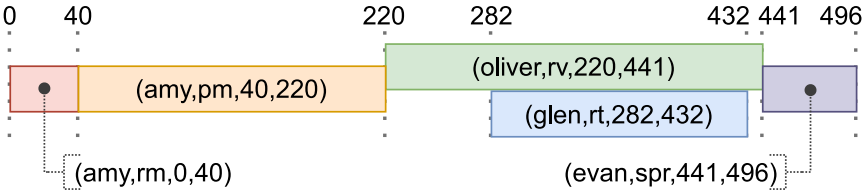
A temporal model as described in Section 2.3:

- `defActDuration`(a, d): the default duration of activity a is estimated as d (i.e., $\Delta(a) = d$);

[4] Renewable resources are available with a constant amount in each time period, e.g., human resources.

```
% selection of a makespan value
makespanDomain(U) :- upperBound(U).                                                                    12
makespanDomain(M1) :- makespanDomain(M), M1=M-1, M1>=0.                                                13
1<={makespan(M) : makespanDomain(M)}<=1.                                                               14

% time domain generation from the makespan
time(0).                                                                                               15
time(T1) :- time(T), T1=T+1, T1<=U, makespan(U).                                                       16

% senior roles can execute activities of junior roles
alAC(A,L1) :- llAC(L1,L2), alAC(A,L2).                                                                 17

% resource-activity duration interval preference
defRAD(R,A,D) :- raDuration(R,A,D), rlAC(R,L), alAC(A,L).                                              18
defRAD(R,A,D) :- not raDuration(R,A,_), laDuration(L,A,D), rlAC(R,L), alAC(A,L).                       19
defRAD(R,A,D) :- not raDuration(R,A,_), not laDuration(L,A,_), defActDuration(A,D), rlAC(R,L), alAC(A,L).  20

% (c.1)
1<={allocation(R,A,S,C): time(S), time(C), defRAD(R,A,D), C=S+D}<=1 :- activity(A).                     21

% (c.2)
:- allocation(_,A,S1,_), allocation(_,A,S2,_), S1<S2.                                                  22

% (c.3)
:- prec(A1,A2), allocation(_,A1,_,C1), allocation(_,A2,S2,_), C1>S2.                                   23

% (c.4)
:- conc(A1,A2), allocation(R,A1,S1,_), allocation(R,A2,S2,C2), S2<=S1, C2>S1, A1<A2.                   24
:- conc(A1,A2), allocation(R,A1,_,C1), allocation(R,A2,S2,C2), S2<C1,  C2>=C1, A1<A2.                  25
:- conc(A1,A2), allocation(R,A1,S1,C1), allocation(R,A2,S2,C2), S2>S1,  C2<C1, A1<A2.                  26

% minimization of the makespan
:~ makespan(M). [M,M]                                                                                  27
```

Fig. 3. ASP encoding for the RABP problem.



OPT: allocation(amy,rm,0,40; amy,pm,40,220; glen,rt,282,432; oliver,rv,220,441; evan,spr,441,496)

Fig. 4. Optimal solution of the book publishing example in Section 2.

- `raDuration`(r, a, d): the duration of activity a is estimated as d when it is executed by resource r from $\mathcal{L}_\mathcal{P}$ (i.e., $\delta_r(\mathcal{L}_\mathcal{P}, r, a) = d$);
- `laDuration`(l, a, d): the duration of activity a is estimated as d when it is executed by a resource that has role l from $\mathcal{L}_\mathcal{P}$ (i.e., $\delta_l(\mathcal{L}_\mathcal{P}, l, a) = d$);
- `upperBound`(u): makespan is bounded at u time units.

The code reads as follows. Rules (12–14) select a makespan for the allocation. Rules (15, 16) generate the time domain from the selected makespan. Rule (17) propagates the permissions of activity executions of a junior role to a senior role (i.e., role-to-role RBAC relations). Rules (18–20) implement the resource-activity duration preference handling mechanism described in Definition 6. Rules (21), (22), (23), and (24–26) correspond to the constraints (c.1), (c.2), (c.3), and (c.4) described in Section 2.4, respectively. Finally, Rule (27) minimizes the makespan.

Output Format. For the allocation output we use the following predicate as described in Section 2.4:

- `allocation`(r, a, s, c): a resource r is allocated to activity a at the start time s until the completion time c (i.e., $(r, a, s, c) \in I$).

BPub Example 7. The book publishing example described in Section 2 (the process model from Fig. 1 and Table 1, the organizational model

Table 5
ASP encoding of the book publishing example described in Section 2.

process model	
activity	(rm; pm; rt; rv; spr)
prec	(rm,pm; rm,rt; rm,rv; rm,spr; pm,rt; pm,rv; pm,spr; rt,spr; rv,spr)
conc	(rt,rv; rv,rt)
organizational model	
alAC	(rm,publ; pm,copyEd; rt,copyEd; rv,graphAr; spr,adAsst)
rlAC	(amy,publ; glen,copyEd; drew,copyEd; emily,copyEd; oliver,graphAr; evan,adAsst)
llAC	(publ,copyEd)
temporal model	
defActDuration	(rm,20; pm,180; rt,240; rv,240; spr,30)
raDuration	(amy,rm,40; drew,pm,247; glen,pm,182; drew,rt,186; glen,rt,150; oliver,rv,221; evan,spr,55)
laDuration	(publ,rm,45; copyEd,pm,208; copyEd,rt,171; graphAr,rv,221; adAsst,spr,55)
upperBound	(350)

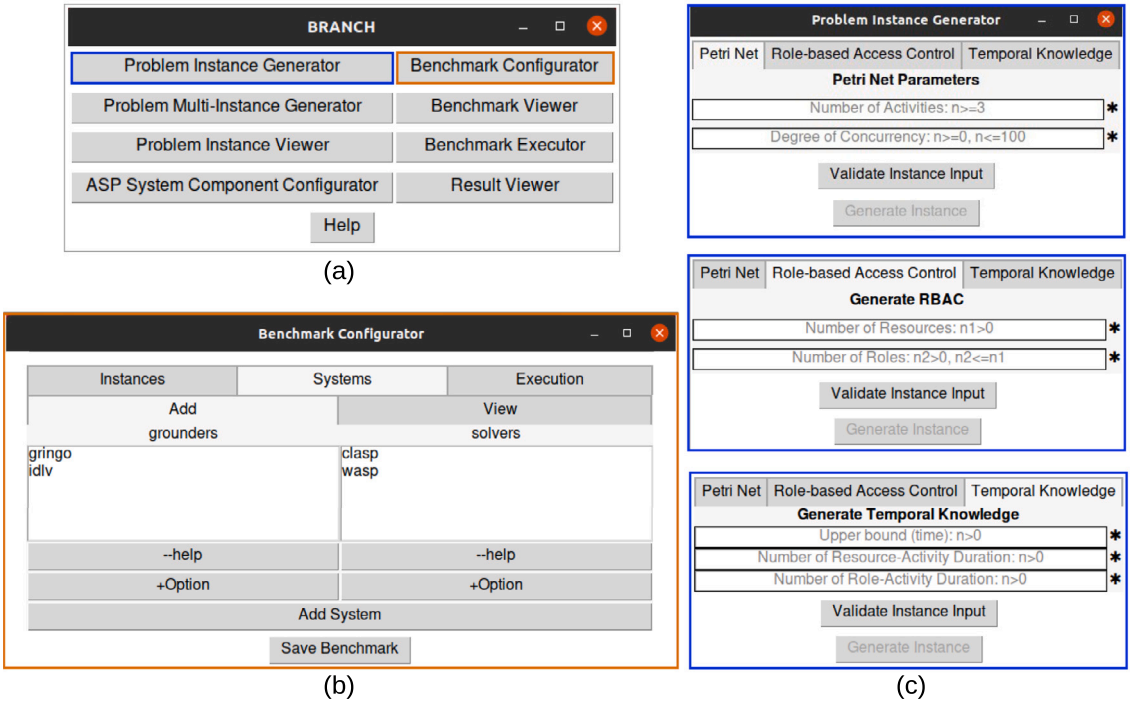


Fig. 5. Screenshots of the benchmark.

Table 6
Parameters for generating one RABP problem instance.

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

from Table 2, and the temporal model from Table 4) is encoded in ASP as depicted in Table 5. Its optimal solution computed by an ASP system using the RABP encoding in Fig. 3 is shown in Fig. 4. The solution can be read as the following: *Amy* is allocated to *Receive Manuscript* at time 0 and it is planned in the resultant allocation that *Amy* takes 40 min to execute this activity. Then, *Amy* is allocated to *Proofread Manuscript* with a starting time of 40 min, and she takes 180 min to execute the activity, and so on.

4. Implementation of the ASP systems benchmark for RABP

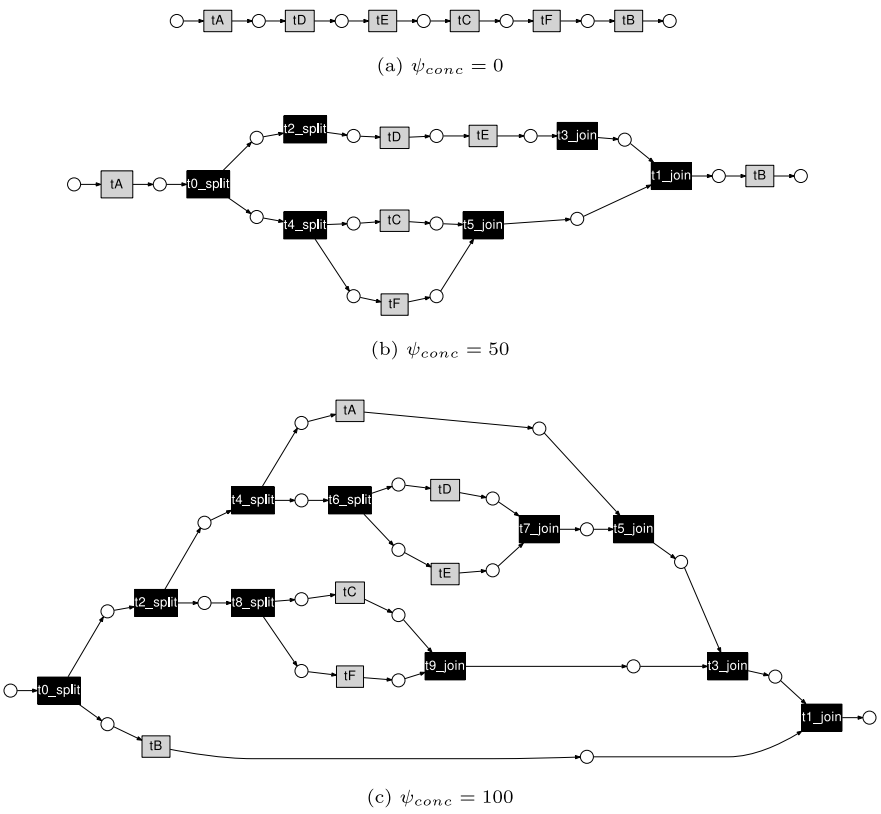
The lack of datasets for benchmarking ASP systems for RABP, unlike other related scheduling domains (Elloumi, Loukil, & Fortemps, 2021; Li & Dong, 2018; Paraskevopoulos, Tarantilis, & Ioannou, 2012), led us to design and implement a ready-to-use benchmark BRANCH that generates RABP problem instances and solves them via user-configured ASP Systems (Havur, Cabanillas, & Polleres, 2021). BRANCH is implemented in a user-friendly fashion with a simple User Interface (UI)[5]. Fig. 5(a) shows the main menu of BRANCH. The functionalities provided within BRANCH are further described below.

Problem (Multi-)Instance Generator: This component involves a process model, an organizational model and a temporal knowledge generator, as depicted in Fig. 5(c). An overview of required parameters for generating one RABP problem instance is summarized in Table 6. The inputs of the process model generator are the number of activities and the degree of concurrency of the generated process model. BRANCH uses the *stochastic Petri net* plug-in of the process mining tool ProM (Rogge-Solti, 2014) for generating Petri nets. This plug-in performs a series of random structured insertions of control-flow constructs resulting in a random Petri net that is sound, free-choice and block-structured (van der Aalst, 1996). For example, Fig. 6 shows three generated Petri nets with different degree of concurrency. After a Petri net is generated, the behavioral relations of its activities are derived for RABP. An organizational model is then generated using the RBAC model (Colantonio et al., 2009) by entering a number of resources and a number of roles that are necessary to create the model. The RBAC model consists of a resource set, a role set, activity-to-role assignment tuples specifying which activity can be executed by the resources associated with which role(s), and tuples of resource-to-role assignments identifying the roles of a resource. Finally, the temporal knowledge is generated given an upper bound for the execution of the process, resource-activity specific durations, and role-activity specific durations (apart from default activity durations). Users can generate multiple problem instances at once via the *multi-instance* generator. Each of the previously described numeric inputs are parameterized by three values (lower limit, mode, and upper limit). Therefore, their respective triangular random variables are sampled to generate multiple RABP instances.

Problem Instance (resp. Benchmark) Viewer: Generated problem instances (resp. benchmarks) are listed in a table including the parameters used for their generation. The user can also remove the instances.

ASP System Component Configurator: The components of the ASP systems are added by naming the component, its type and the executable path. We include the state-of-the-art ASP *grounders* GRINGO (Gebser et al., 2011) and I-DLV (Calimeri et al., 2017); and

[5] The UI is implemented via the Python appJar module.



(a) $\psi_{conc} = 0$

(b) $\psi_{conc} = 50$

(c) $\psi_{conc} = 100$

Fig. 6. Three generated Petri nets with 6 activities.

ASP *solvers* CLASP (Gebser et al., 2015) and WASP (Alviano et al., 2015) in BRANCH for the sake of convenience of the benchmark user. These grounders and solvers are selected mainly due to being among the top-performing tools in the latest ASP Competitions (Gebser et al., 2017, 2020).

Benchmark Configurator: To set up a new benchmark instance the user can select problem instances to include as well as add new ASP systems to be used by selecting and customizing (e.g., adding command line options) grounder+solver configurations with the UI shown in Fig. 5(b). A custom problem encoding for RABP can also be selected if desired and the execution details of the benchmark (e.g., time and memory limitations) can be further constrained.

Benchmark Executor: The user can select one benchmark instance to execute. The grounding and solving instances are dynamically listed in the monitored interface.

Result Viewer: The time and memory usage statistics of ASP grounders and solvers can be exported in *csv* format. We also implemented box plots and cactus plots for an easier interpretation of the benchmark results.

BRANCH follows the applicable basic principles of experimental design by providing (i) *randomization* via the multi-instance generator, and (ii) *replication* via the benchmark configurator and benchmark executor.

The benchmark software, a video that screencasts the tool usage and a separate tutorial document are available at https://urban.ai.wu.ac.at/~havur/eswa2022/.

5. BRANCH in use

As aforementioned, BRANCH is configurable in the sense that any ASP grounder and ASP solver can be added and their optional parameters are textually given for purposes like activating their meta-heuristics and ensuring the compatibility between grounders and solvers, when possible. We demonstrate the functionalities of BRANCH via configuring four different ASP systems using out-of-the-box grounders and solvers: GRINGO+CLASP (i.e., CLINGO; Gebser, Kaminski, Kaufmann, & Schaub, 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 5.5.0, I-DLV 1.1.6, CLASP 3.3.6, WASP 2.0. These tools are executed on their default meta-heuristic options (i.e., no extra parameter is given) in this benchmark except for the combination of the grounder GRINGO and the solver WASP. While configuring the ASP system GRINGO+WASP, we set the output format of GRINGO to *smodels* (Syrjänen, 2000) by simply adding "`-o smodels`" parameter for GRINGO using the `+Option` button in Fig. 5(b) because GRINGO's default output format *aspif* (Gebser et al., 2016) is not compatible with WASP.

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

Problem Instances: We generated 70 problem instances using BRANCH's instance generator. The details of these instances are provided in Table 7. 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, ψ_{conc} is the degree of concurrency 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 the given upper bound u.

We ran the benchmark and summarized the results in Tables 8 and 9. Table 8 shows the performance statistics of two grounders while grounding the problem instances. In the table, *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 9 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.

Table 7
Properties of problem instances.

id	n_A	δ_{conc}	n_R	n_L	SAT	u	id	n_A	δ_{conc}	n_R	n_L	SAT	u	id	n_A	δ_{conc}	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

Table 8
Grounder statistics.

id	GRINGO			I-DLV			id	GRINGO			I-DLV		
	time	*mem*	$\|g(\Pi)\|$	*time*	*mem*	$\|g(\Pi)\|$		*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

c_{max} is the makespan (i.e., maximum of the activity completion times) computed by the system, which is minimized by the weak constraint. Within this column, bold and italic numbers are the confirmed optima cases; and 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 tables "–" within the *time* column means "out-of-time" and "–" within the *mem* column means "out-of-memory".

The box plots in Fig. 7 visually summarize Tables 8 and 9. 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

Table 9
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	

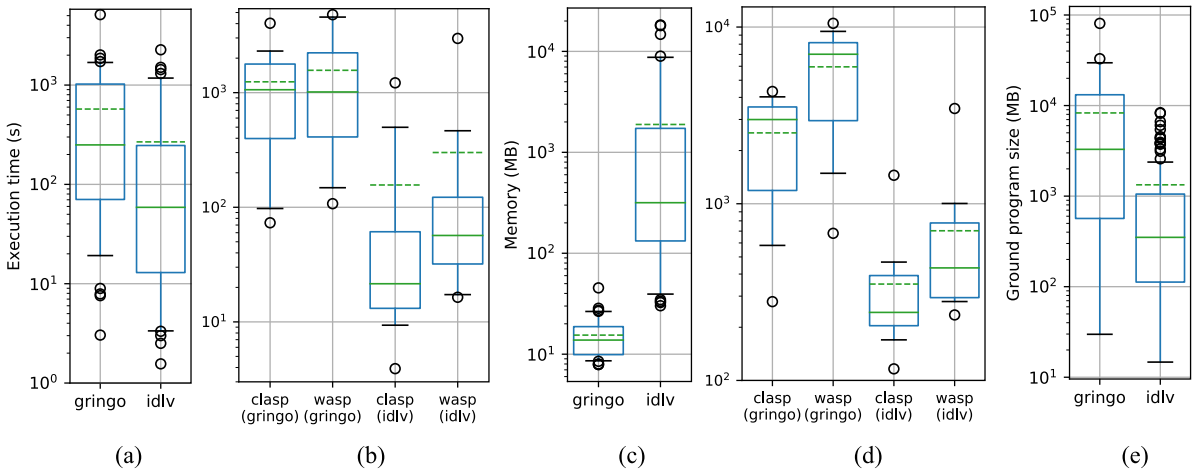


Fig. 7. Box plots regarding the performance statistics of grounders and solvers.

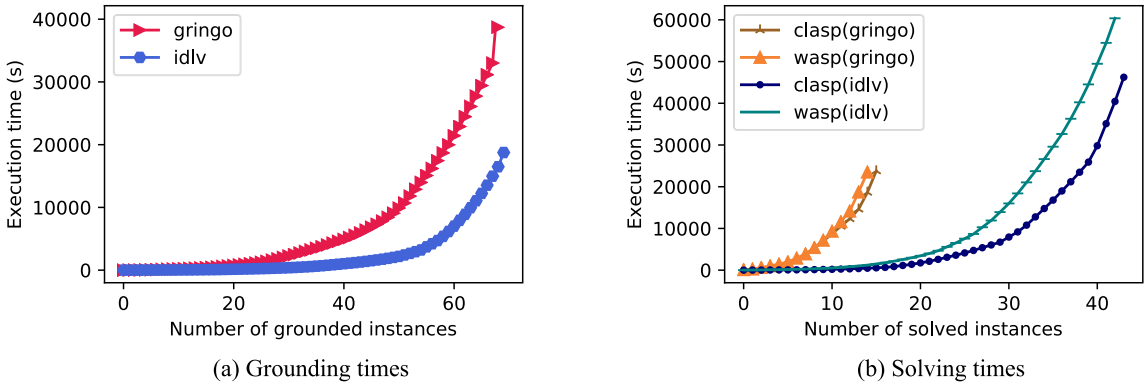


Fig. 8. Sorted cactus plots.

represents the *mean* of the sample. Fig. 7(a) and Fig. 7(b) compare the CPU execution times of grounders and solvers. Fig. 7(c) and Fig. 7(d) compare the memory usage of grounders and solvers. Fig. 7(e) reports on the ground program sizes of the instances. By looking at Fig. 7(a) and Fig. 7(c), we find out that I-DLV grounds the problem instances much faster than GRINGO, although 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 much smaller than those generated by GRINGO (cf. Table 8 and Fig. 7(e)).

The cactus plots in Fig. 8 separately compare the grounder and solver performances. Less steep curves (cf. Fig. 8(b)) and smaller memory footprint (cf. Fig. 7(d)) of CLASP(GRINGO) and CLASP(I-DLV) – in comparison to those of WASP(GRINGO) and WASP(I-DLV) – illustrate that the ASP solver CLASP performs better than WASP. To address the makespan minimization, both solvers contain several solving methods: model-guided methods aim to produce models with descending costs, and core-guided methods identify and relax unsatisfiable cores until a model is found (Alviano et al., 2015; Gebser et al., 2015). In the default setting, CLASP uses its branch-and-bound algorithm (Gebser, Kaufmann, & Schaub, 2012), while WASP utilizes a core-guided algorithm (Saikko, Dodaro, Alviano, & Järvisalo, 2018). Results suggest that both solvers behave non-optimally, mainly because the given ASP encoding for the RABP problem is rather tailored towards a declarative and readable set of rules than for a performance gain of particular underlying solver techniques. Therefore, we see room for improvement not only in solver techniques but also in bespoke problem encoding optimizations, as they have been successfully applied to other ASP benchmark problems in the past (Gebser et al., 2020).

Overall, I-DLV+CLASP completes 41 instances within the given time and memory constraints whereas I-DLV+WASP, GRINGO+CLASP, and GRINGO+WASP complete 40, 16, and 15 instances, respectively. This result shows that I-DLV+CLASP is the most performant ASP system of our baseline benchmark for RABP.

6. Limitations

The RABP problem is characterized by the essential ideas behind designing and managing business processes and organizational models (Dumas et al., 2018; Rosemann & vom Brocke, 2015; Rummler & Ramias, 2015). Problems similar to RABP have been widely acknowledged in research areas other than Logic Programming (LP), such as Constraint Programming (CP), Machine Learning (ML) and Operations Research (OR) (Bouajaja & Dridi, 2017; Lombardi & Milano, 2012). The survey conducted in (Pufahl, Ihde, Stiehle, Weske, & Weber, 2021) presents that automatic resource allocation in business processes has a wide variety of implementations encompassing a subset of capabilities for:

- **allocation mode** (one resource to one activity, one resource to many activities, many resources to one activity, and many resources to many activities),
- **optimization goal** (finding best-fitting resource, makespan minimization, cost minimization, balancing workload among the resources, etc.),
- **resource taxonomies and attributes** (previous performance, workload, role, expertise, etc.), and

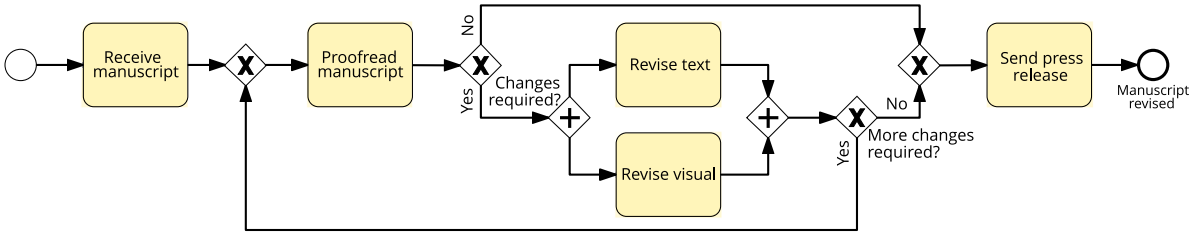


Fig. 9. BPMN model of the book publishing process with decision nodes.

- **type of evaluation** (simulation experiments, experiments with real-world data, case study, etc.).

This survey also shows the statistics of the studies that focus on these capabilities. To find the right balance in the complexity of the problem definition and to keep the focus on the ASP system benchmark BRANCH, we implemented the most studied capabilities: allocation of *one resource to one activity*, *finding best-fitting resource*, *makespan minimization*, *role-based resource taxonomy* (i.e., RBAC model), and *performance-based resource attributes* (resource- and role-based resource-activity durations) as described in Section 2. In our previous work, we also addressed resource allocation with extended capabilities, namely, a Petri net firing semantics-based resource allocation (Havur et al., 2015) that *simulates* the behavior of the Petri net, and an extended encoding that includes further requirements (Havur et al., 2016), such as allocation of *many resources to one activity*, allocation of *non-human resources*, *expertise* levels of resources, and *cost minimization*. Such capabilities could also be included in BRANCH with a reasonable degree of effort, provided that ASP encodings are already available.

Fig. 9 depicts the BPMN model of the book publishing process including decision points. The main difference between the running example model in Fig. 1 and this model is that, in this process model, after proofreading the manuscript, *if changes are required* the modifications suggested must be applied on text and figures. This review-and-improvement *loop* is *repeated* until there are no more changes to apply. In real-world scenarios, such decision points are crucial for describing the processes. However, this kind of uncertainty in the execution time of processes causally divides the search space for RABP (i.e., the complete set of activities to execute RABP cannot be known before execution). To address this issue, first, a decision-node-free process fragment (i.e., partial-run) needs to be generated under a number of assumptions on these decision nodes. For example, given the book publishing process including the two decision points in Fig. 9, when it is assumed that *changes required* at the first decision node (the branch labeled with *yes*) is taken, and *no more changes are required* at the second decision node (the branch labeled with *no*) is taken, the decision-node-free process fragment in Fig. 1 is generated. In case an assumption does not hold at run time, a new decision-node-free process fragment (only from the decision point onward) is generated, and a new RABP instance is triggered (i.e., resource reallocation). Our previous work investigated the most effective and robust way of generating decision-node-free process fragments from process models with decision nodes for RABP (Havur & Cabanillas, 2019). As RABP cannot be performed when there is a decision node in the process, we omit the decision nodes in the formalization of RABP in Section 2, and also in the problem instance generator in Section 4.

7. Conclusions

We have formalized the RABP problem and provided a new benchmark for ASP systems. The results of the illustrative benchmark run show that RABP is a challenging problem for the ASP systems that are among the most performant in the previous ASP Competitions (Gebser et al., 2017, 2020). This application-oriented benchmark would be beneficial to the ASP community by helping assess advances in the formalism (e.g., the ease of encoding – i.e., the compactness, readability, modularity and maintainability of the problem encoding) and the computational performance of the solvers; and further, encourage the BPM community to integrate and test RABP in the process execution environments.

Future work will involve optimizing the ASP encoding of RABP to improve its computational efficiency to operate in large-scale real-world scenarios. It is also on our agenda to extend BRANCH towards more flexible grounding and solving paradigms such as *multi-shot ASP solving* (Gebser, Kaminski, Kaufmann, & Schaub, 2019) (e.g., to support the use of an incremental encoding of RABP) and formalisms other than ASP. Moreover, devising an interface to existing BPMSs from BRANCH might prove useful in building and deploying tailored solutions for resource allocation needs in BPM.

CRediT authorship contribution statement

Giray Havur: Conceptualization, Methodology, Software, Validation, Visualization, Writing – original draft. **Cristina Cabanillas:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Axel Polleres:** Conceptualization, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by: (i) European Regional Development Fund (ERDF)-A way of making Europe, (ii) CONFLEX project (grant number RTI2018-100763-J-I00) funded by Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación (MCIN/AEI/10.13039/501100011033), and (iii) MEMENTO project (grant number US-1381595) funded by Programa Operativo FEDER 2014-2020 and Junta de Andalucía (Consejería de Economía, Conocimiento, Empresas y Universidad). Axel Polleres' work is supported by TEAMING.AI project (grant number 957402) funded by the European Union's Horizon 2020 research and innovation program.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2022.117599.

References

Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., et al. (2017). The ASP system DLV2. In M. Balduccini, T. Janhunen (Eds.), *Lecture notes in computer science*: *vol. 10377*, *Proceedings of the 14th international conference on logic programming and nonmonotonic reasoning* (pp. 215–221). Springer, http://dx.doi.org/10.1007/978-3-319-61660-5_19.

Alviano, M., Dodaro, C., Leone, N., & Ricca, F. (2015). Advances in WASP. In F. Calimeri, G. Ianni, & M. Truszczynski (Eds.), *Lecture notes in computer science*: *vol. 9345*, *Proceedings of the 13th international conference on logic programming and nonmonotonic reasoning* (pp. 40–54). Springer, http://dx.doi.org/10.1007/978-3-319-23264-5_5.

Bouajaja, S., & Dridi, N. (2017). A survey on human resource allocation problem and its applications. *Operational Research*, *17*(2), 339–369. http://dx.doi.org/10.1007/s12351-016-0247-8.

Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92–103. http://dx.doi.org/10.1145/2043174.2043195.

Buccafurri, F., Leone, N., & Rullo, P. (1997). Adding weak constraints to disjunctive datalog. In M. Falaschi, M. Navarro, & A. Policriti (Eds.), *Proceedings of the joint conference on declarative programming* (pp. 557–568).

Burattin, A. (2015). Introduction to business processes, BPM, and BPM systems. In *Process mining techniques in business environments: Theoretical aspects, algorithms, techniques and open challenges in process mining* (pp. 11–21). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-17482-2_2.

Cabanillas, C., Resinas, M., del-Río-Ortega, A., & Cortés, A. R. (2015). Specification and automated design-time analysis of the business process human resource perspective. *Information Systems*, *52*, 55–82. http://dx.doi.org/10.1016/j.is.2015.03.002.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., et al. (2020). ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, *20*(2), 294–309. http://dx.doi.org/10.1017/S1471068419000450.

Calimeri, F., Fuscà, D., Perri, S., & Zangari, J. (2017). I-DLV: The new intelligent grounder of DLV. *Intelligenza Artificiale*, *11*(1), 5–20. http://dx.doi.org/10.3233/IA-170104.

Colantonio, A., Pietro, R. D., Ocello, A., & Verde, N. V. (2009). A formal framework to elicit roles with business meaning in RBAC systems. In B. Carminati, & J. Joshi (Eds.), *Proceedings of the 14th ACM symposium on access control models and technologies* (pp. 85–94). ACM, http://dx.doi.org/10.1145/1542207.1542223.

Colmerauer, A., & Roussel, P. (1993). The birth of Prolog. In J. A. N. Lee, & J. E. Sammet (Eds.), *Preprints of the 2nd ACM SIGPLAN conference on History of programming languages conference* (pp. 37–52). ACM, http://dx.doi.org/10.1145/154766.155362.

Denecker, M., Vennekens, J., Bond, S., Gebser, M., & Truszczynski, M. (2009). The second answer set programming competition. In E. Erdem, F. Lin, & T. Schaub (Eds.), *Proceedings of the 10th international conference on logic programming and nonmonotonic reasoning* (pp. 637–654). Springer, http://dx.doi.org/10.1007/978-3-642-04238-6_75.

Dijkman, R. M., Dumas, M., & Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. *Information and Software Technology*, *50*(12), 1281–1294. http://dx.doi.org/10.1016/j.infsof.2008.02.006.

Drescher, C., Gebser, M., Kaufmann, B., & Schaub, T. (2010). Heuristics in conflict resolution. *The Computing Research Repository*, http://dx.doi.org/10.48550/arXiv.1005.1716, (CoRR), arXiv.

Dumas, M., Rosa, M. L., Mendling, J., & Reijers, H. A. (2018). *Fundamentals of business process management* (2nd ed.). Springer, http://dx.doi.org/10.1007/978-3-662-56509-4.

Elloumi, S., Loukil, T., & Fortemps, P. (2021). Reactive heuristics for disrupted multi-mode resource-constrained project scheduling problem. *Expert Systems with Applications*, *167*, Article 114132. http://dx.doi.org/10.1016/j.eswa.2020.114132.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, *1*(2), 117–129. http://dx.doi.org/10.1287/moor.1.2.117.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Wanko, P. (2016). Theory solving made easy with clingo 5. In M. Carro, A. King, N. Saeedloei, & M. D. Vos (Eds.), *OASIcs*: *vol. 52*, *Technical communications of the 32nd international conference on logic programming* (pp. 2:1–2:15). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, http://dx.doi.org/10.4230/OASIcs.ICLP.2016.2.

Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., & Schaub, T. (2015). Progress in clasp series 3. In F. Calimeri, G. Ianni, & M. Truszczynski (Eds.), *Lecture notes in computer science*: *vol. 9345*, *Proceedings of the 13th international conference on logic programming and nonmonotonic reasoning* (pp. 368–383). Springer, http://dx.doi.org/10.1007/978-3-319-23264-5_31.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer set solving in practice*. Morgan & Claypool Publishers, http://dx.doi.org/10.2200/S00457ED1V01Y201211AIM019.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). Clingo = ASP + Control: Preliminary report. *The Computing Research Repository*, http://dx.doi.org/10.48550/arxiv.1405.3694, (CoRR), arXiv.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2019). Multi-shot ASP solving with clingo. *Theory Practice of Logic Programming*, *19*(1), 27–82. http://dx.doi.org/10.1017/S1471068418000054.

Gebser, M., Kaminski, R., König, A., & Schaub, T. (2011). Advances in gringo series 3. In J. P. Delgrande, & W. Faber (Eds.), *Lecture notes in computer science*: *vol. 6645*, *Proceedings of the 11th international conference on logic programming and nonmonotonic reasoning* LPNMR 2011, (pp. 345–351). Springer, http://dx.doi.org/10.1007/978-3-642-20895-9_39.

Gebser, M., Kaminski, R., Ostrowski, M., Schaub, T., & Thiele, S. (2009). On the input language of ASP grounder gringo. In E. Erdem, F. Lin, & T. Schaub (Eds.), *Proceedings of the 10th international conference on logic programming and nonmonotonic reasoning* (pp. 502–508). Springer, http://dx.doi.org/10.1007/978-3-642-04238-6_49.

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187*, 52–89. http://dx.doi.org/10.1016/j.artint.2012.04.001.

Gebser, M., Maratea, M., & Ricca, F. (2017). The sixth answer set programming competition. *Journal of Artificial Intelligence Research*, *60*, 41–95. http://dx.doi.org/10.1613/jair.5373.

Gebser, M., Maratea, M., & Ricca, F. (2020). The seventh answer set programming competition: Design and results. *Theory and Practice of Logic Programming*, *20*(2), 176–204. http://dx.doi.org/10.1017/S1471068419000061.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In R. A. Kowalski, & K. A. Bowen (Eds.), *Proceedings of the 5th international conference and symposium on logic programming* (pp. 1070–1080). MIT Press.

Havur, G., & Cabanillas, C. (2019). History-aware dynamic process fragmentation for risk-aware resource allocation. In H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, & R. Meersman (Eds.), *Proceedings of OTM 2019 conferences - confederated international conferences: CoopIS, ODBASE, C&TC 2019* (pp. 533–551). http://dx.doi.org/10.1007/978-3-030-33246-4_33.

Havur, G., Cabanillas, C., Mendling, J., & Polleres, A. (2015). Automated resource allocation in business processes with answer set programming. In M. Reichert, & H. A. Reijers (Eds.), *Revised papers of the 13th international business process management workshops* (pp. 191–203). Springer, http://dx.doi.org/10.1007/978-3-319-42887-1_16.

Havur, G., Cabanillas, C., Mendling, J., & Polleres, A. (2016). Resource allocation with dependencies in business process management systems. In M. L. Rosa, P. Loos, & O. Pastor (Eds.), *Lecture notes in business information processing*: *vol. 260*, *Proceedings of the business process management Forum* BPM Forum 2016, (pp. 3–19). Springer, http://dx.doi.org/10.1007/978-3-319-45468-9_1.

Havur, G., Cabanillas, C., & Polleres, A. (2021). BRANCH: An ASP systems benchmark for resource allocation in business processes. In W. M. P. van der Aalst, R. M. Dijkman, A. Kumar, F. Leotta, F. M. Maggi, J. Mendling, B. T. Pentland, A. Senderovich, M. Sepúlveda, E. S. Asensio, & M. Weske (Eds.), *CEUR workshop proceedings*: *vol. 2973*, *Proceedings of the best dissertation award, doctoral consortium, and demonstration & resources track at BPM 2021* (pp. 176–180). CEUR-WS.org, URL http://ceur-ws.org/Vol-2973/paper_285.pdf.

Horling, B., & Lesser, V. R. (2004). A survey of multi-agent organizational paradigms. *Knowledge Engineering Review*, *19*(4), 281–316. http://dx.doi.org/10.1017/S0269888905000317.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., et al. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, *7*(3), 499–562. http://dx.doi.org/10.1145/1149114.1149117.

Li, H., & Dong, X. (2018). Multi-mode resource leveling in projects with mode-dependent generalized precedence relations. *Expert Systems with Applications*, *97*, 193–204. http://dx.doi.org/10.1016/j.eswa.2017.12.030.

Lifschitz, V. (2008). What is answer set programming? In D. Fox, & C. P. Gomes (Eds.), *Proceedings of the 23rd AAAI conference on artificial intelligence* (pp. 1594–1597). AAAI Press, URL http://www.aaai.org/Library/AAAI/2008/aaai08-270.php.

Lombardi, M., & Milano, M. (2012). Optimal methods for resource allocation and scheduling: A cross-disciplinary survey. *Constraints*, *17*(1), 51–85. http://dx.doi.org/10.1007/s10601-011-9115-6.

OMG (2014). Business process model and notation (BPMN). [Specification], https://www.omg.org/spec/BPMN/2.0.2/PDF.

Paraskevopoulos, D. C., Tarantilis, C. D., & Ioannou, G. (2012). Solving project scheduling problems with resource constraints via an event list-based evolutionary algorithm. *Expert Systems with Applications*, *39*(4), 3983–3994. http://dx.doi.org/10.1016/j.eswa.2011.09.062.

Peterson, J. L. (1981). *Petri net theory and the modeling of systems*. Englewood Cliffs, N.J., Prentice-hall.

Pufahl, L., Ihde, S., Stiehle, F., Weske, M., & Weber, I. (2021). Automatic resource allocation in business processes: A systematic literature survey. *The Computing Research Repository*, (CoRR), arXiv https://arxiv.org/abs/2107.07264.

Rogge-Solti, A. (2014). Stochastic petri net plug-in of the process mining framework prom. [Computer Software], http://www.promtools.org/.

Rosemann, M., & vom Brocke, J. (2015). The six core elements of business process management. In J. vom Brocke, & M. Rosemann (Eds.), *International handbooks on information systems*, *Handbook on business process management 1: Introduction, methods, and information systems* (2nd ed.). (pp. 105–122). Springer, http://dx.doi.org/10.1007/978-3-642-45100-3_5.

Rummler, G. A., & Ramias, A. J. (2015). A framework for defining and designing the structure of work. In J. vom Brocke, & M. Rosemann (Eds.), *International handbooks on information systems*, *Handbook on business process management 1: Introduction, methods, and information systems* (2nd ed.). (pp. 81–104). Springer, http://dx.doi.org/10.1007/978-3-642-45100-3_4.

Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Edmond, D. (2005). Workflow resource patterns: Identification, representation and tool support. In O. Pastor, & J. F. e Cunha (Eds.), *Lecture notes in computer science*: *vol. 3520*, *Proceedings of the 17th international conference on advanced information systems engineering* (pp. 216–232). Springer, http://dx.doi.org/10.1007/11431855_16.

Saikko, P., Dodaro, C., Alviano, M., & Järvisalo, M. (2018). A hybrid approach to optimization in answer set programming. In M. Thielscher, F. Toni, & F. Wolter (Eds.), *Proceedings of the 16th international conference on principles of knowledge representation and reasoning* (pp. 32–41). AAAI Press, URL https://aaai.org/ocs/index.php/KR/KR18/paper/view/18021.

Syrjänen, T. (2000). *Lparse 1.0 user's manual*. URL http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.

van der Aalst, W. M. P. (1996). *Structural characterizations of sound workflow nets*: *Computing science reports, Vol. 9623*, Technische Universiteit Eindhoven, https://pure.tue.nl/ws/files/2454992/9712195.pdf.

van der Aalst, W. M. P. (2016). *Process mining - Data science in action* (2nd ed.). Springer, http://dx.doi.org/10.1007/978-3-662-49851-4.

van der Aalst, W. M. P., & van Dongen, B. F. (2013). Discovering Petri nets from event logs. *Transactions on Petri Nets and Other Models of Concurrency*, *7*, 372–422. http://dx.doi.org/10.1007/978-3-642-38143-0_10.

Weidlich, M., Mendling, J., & Weske, M. (2011). Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering*, *37*(3), 410–429. http://dx.doi.org/10.1109/TSE.2010.96.