BRANCH: An ASP Systems Benchmark for Resource Allocation in Business Processes*

Giray Havur[1,3], Cristina Cabanillas[2] and Axel Polleres[1]

[1]Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria
[2]Universidad de Sevilla, Avda. Reina Mercedes s/n, ETS Ingeniería Informática, 41012 Seville, Spain
[3]Siemens AG Austria, Technology, Siemensstraße 90, 1210 Vienna, Austria

Abstract
The goal of BRANCH is to benchmark Answer Set Programming (ASP) systems to test their performance when dealing with the task of automatically allocating resources to business process activities. Like many other scheduling problems, the allocation of resources and starting times to process activities is a challenging optimization problem, yet it is a crucial step for an optimal execution of the processes. BRANCH has been designed as a configurable benchmark equipped with instance generators that produce problem instances of different size and hardness with respect to adjustable parameters. This application-oriented benchmark supports the BPM community to find the ASP systems and implementations that perform better in solving the resource allocation problem.

Keywords
answer set programming, benchmark, resource allocation

1. Introduction and Significance to BPM

As an integral part of business processes, resources have to be considered throughout all the stages of the Business Process Management (BPM) lifecycle, which iterates from process discovery and modeling to process execution and monitoring targeting continuous process improvement [1]. 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 predefined 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 information. Therefore, 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 the resource assignment

Proceedings of the Demonstration & Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM 2021 co-located with the 19th International Conference on Business Process Management, BPM 2021, Rome, Italy, September 6–10, 2021
ghavur@wu.ac.at (G. Havur); cristinacabanillas@us.es (C. Cabanillas); axel.polleres@wu.ac.at (A. Polleres)
0000-0002-6898-6166 (G. Havur); 0000-0001-9182-8847 (C. Cabanillas); 0000-0001-5670-1146 (A. Polleres)

*This work has been partially supported by FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación under project CONFLEX (RTI2018-100763-J-I00).

constraints) and assigned to the task. The (pre-)execution planning of such assignments and their scheduling is addressed in *resource allocation in business processes* (RABP) [2].

The complexity of RABP arises from coordinating the dependencies across a broad set of resources and process activities as well as from solving potential conflicts (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) [3]. 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 [1]. Typically, a subset of such objectives is selected and each one in this subset is assigned a relative priority that defines the respective optimization function.

The complexity and practical importance of this issue has prompted BPM researchers to develop a number of approaches for automatic resource allocation with different optimization functions [2]. One way of addressing RABP is by describing it in a knowledge representation formalism. Answer Set Programming (ASP) [4] has been used to solve various hard computational problems and proved to maintain a balance between expressiveness, ease of use and computational effectiveness [4]. An ASP program is first grounded by a *grounder* to a finite set of clauses where each rule with variables is instantiated as equivalent propositional rules without variables. Afterwards, a *solver* searches for answer sets (i.e. solutions) of the ground program. In the absence of complete information, *default behaviours* related to an allocated resource (e.g. assumed/default duration of a generic activity that may be potentially overridden by a known different duration for a specific resource/role) can be elegantly represented in ASP. Thanks to its flexible modeling constructs, in prior research efforts [5, 3] we demonstrated the suitability of ASP 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; however, so far a comprehensive set of realistic benchmark instances is missing to stress-test the performance of ASP systems in practice on this class of problems.

Driven by the relevance of RABP for both BPM and ASP researchers and practitioners, we have developed BRANCH. In a nutshell, BRANCH provides *problem instance generators* to ensure all the input required for RABP, a *declarative baseline ASP encoding* for the problem and *configuration capabilities* such that different implementations and combinations of ASP grounders and solvers can be benchmarked. The BPM community will benefit from a new tool with a User Interface (UI) to test different ASP systems for RABP. In addition, given that one of the reasons for the limited support for resource allocation in BPM is the lack of existing, openly available instance data to develop/test new approaches (as companies tend to consider resource information sensitive), the built-in problem instance generators shall help to fill this gap.

2. Tool Description

BRANCH has been implemented in a user-friendly fashion with a simple UI[2]. Fig. 1(a) shows the main menu. Its principal functionalities are outlined next.

Problem (Multi-)Instance Generator: It consists of a process model, an organizational model

[1]The makespan is the temporal distance from the start to the end of a process execution.

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

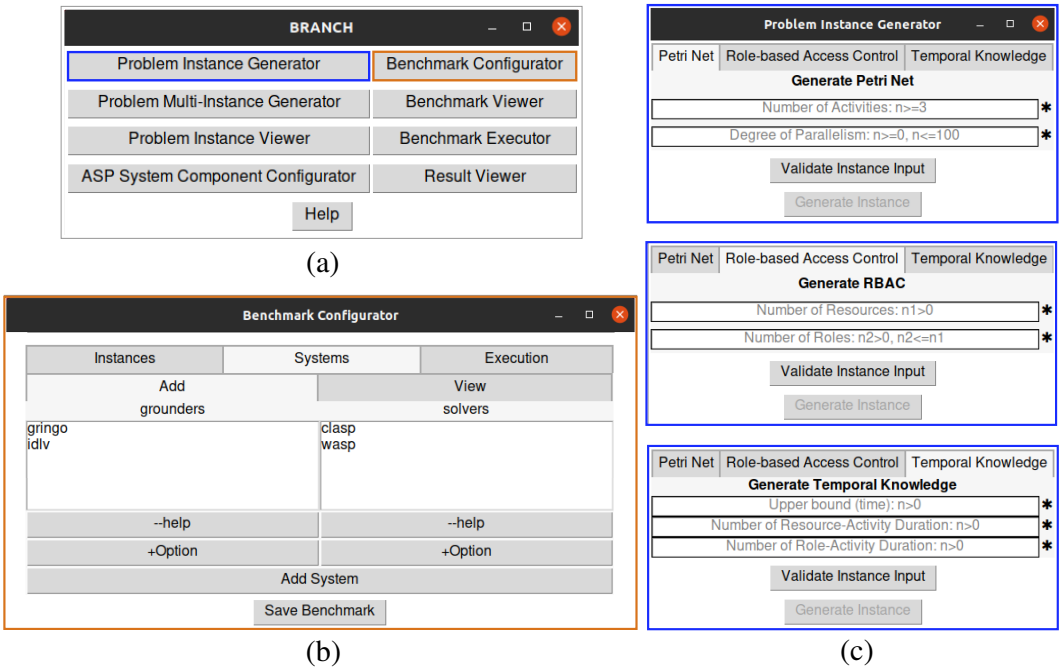

Figure 1: Screenshots of BRANCH

and a temporal knowledge generator, as depicted in Fig. 1(c). The inputs of the process model generator are the number of activities and the degree of parallelism of the generated process model. BRANCH uses the *Generate block-structured stochastic Petri net* plug-in [6] of the process mining tool ProM [7]. It performs a series of random structured insertion of control-flow constructs resulting in a random Petri net that is sound, free-choice and block-structured [8]. An organizational model is then generated using the Role-Based Access Control (RBAC) model [9] 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, a number of resource-activity specific durations, and a number of 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 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 executable path. BRANCH includes the state-of-the-art ASP *grounders* GRINGO and I-DLV, and ASP *solvers* CLASP and WASP.

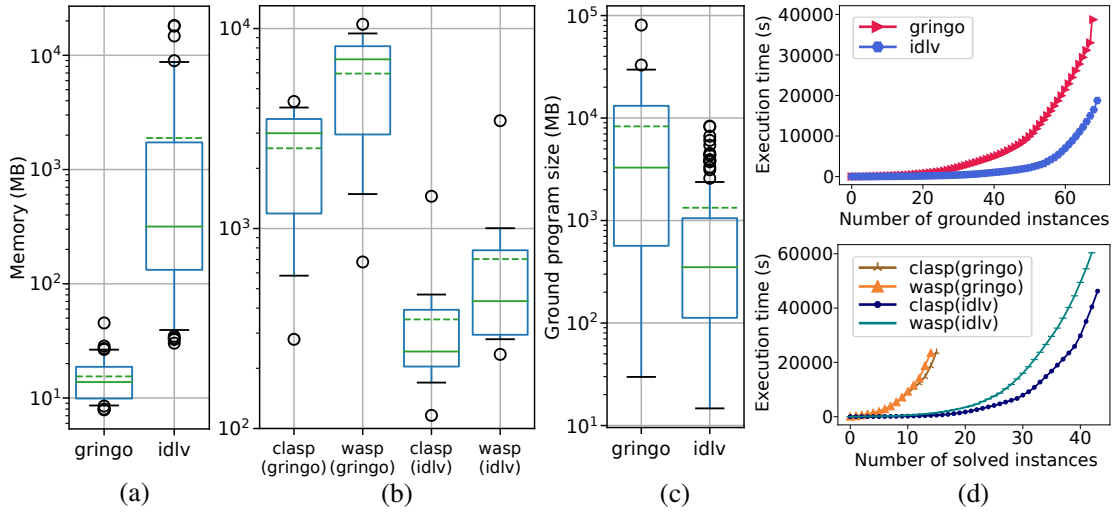

Figure 2: Memory usage statistics of (a) grounders and (b) solvers, (c) ground program sizes of different grounders, and (d) time usage statistics of grounders and solvers

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 grounder+solver configurations with the UI shown in Fig. 1(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: User can select one benchmark instance to execute. The grounding and solving instances are dynamically listed in the interface monitored.

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. The box plots in Fig. 2 report the memory usage statistics of ASP grounders and solvers of a previously run benchmark[3]. Therein, the 90% of the samples are between the upper and lower whiskers. The solid green line represents the *median*, and the dashed green line represents the *mean* of the sample. Fig. 2(a) and Fig. 2(b) compare the memory usage of grounders and solvers. Fig. 2(c) reports on the ground program sizes of the instances. The cactus plots in Fig.2(d) separately compare the grounder and solver time performances. Less steep curves (cf. Fig. 2(d), bottom) and the smaller memory footprint (cf. Fig. 2(a)) of CLASP(GRINGO) and CLASP(I-DLV) – compared to those of WASP(GRINGO) and WASP(I-DLV) – indicates that the ASP solver CLASP performs better than WASP.

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 and a separate tutorial document are available at https://urban.ai.wu.ac.at/~havur/bpmdemo2021/.

[3]Our baseline benchmark run comprises 4 different ASP Systems (i.e. grounder+solver): GRINGO+CLASP, GRINGO+WASP, I-DLV+CLASP, and I-DLV+WASP; and 70 problem instances. The (lower, mode, upper) values for the problem instance generation are: number of activities (4, 40, 80), number of resources (2, 20, 80), number of roles (1, 10, 40), upper bound (50, 500, 1000). Time/memory limit per problem instance run is 7200s/16GB.

3. Maturity and Future Work

BRANCH emerged after a formalization of the RABP problem and stable implementations with ASP [5, 3]. These implementations were tested with real data provided by a partner company as well as in the SHAPEworks tool [10]. The tests performed showed that RABP is challenging for the current ASP systems that received the highest performance scores in former ASP Programming Competitions [11]. The problem instance generator incorporated in BRANCH is capable of generating problem instances in real-world sizes which pose a challenge on state-of-the-art ASP systems.

Future optimizations of the ASP problem encoding to improve its computational efficiency might increase the applicability of BRANCH. The benchmark could also be extended to enable the comparison of other formalisms. Such extensions might prove quite beneficial to both the logic programming and the BPM communities.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management (Second Edition), Springer, 2018. doi:10.1007/978-3-662-56509-4.

[2] C. Cabanillas, Process-and Resource-Aware Information Systems, in: Int. Enterprise Distributed Object Computing Conf. (EDOC), IEEE Computer Society, 2016, pp. 1–10. doi:10.1109/EDOC.2016.7579383.

[3] G. Havur, C. Cabanillas, J. Mendling, A. Polleres, Resource Allocation with Dependencies in Business Process Management Systems, in: Int. Conf. on Business Process Management (BPM) - Forum, volume 260, Springer, 2016, pp. 3–19.

[4] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer set solving in practice, Synthesis Lectures on Artificial Intelligence and Machine Learning 6 (2012) 1–238.

[5] G. Havur, C. Cabanillas, J. Mendling, A. Polleres, Automated Resource Allocation in Business Processes with Answer Set Programming, in: BPM Workshops (BPI), volume 256, Springer, 2015, pp. 191–203.

[6] A. Rogge-Solti, Block-structured stochastic Petri net generator (ProM plug-in), http://www.promtools.org/, 2014. Accessed: 2021-21-06.

[7] W. M. P. van der Aalst, Process Mining - Data Science in Action (Second Edition), Springer, 2016. doi:10.1007/978-3-662-49851-4.

[8] W. M. van der Aalst, Structural characterizations of sound workflow nets, Computing Science Reports 96 (1996) 18–22.

[9] A. Colantonio, R. Di Pietro, A. Ocello, N. V. Verde, A formal framework to elicit roles with business meaning in RBAC systems, in: ACM symposium on Access control models and technologies (SACMAT), ACM, 2009, pp. 85–94.

[10] S. Bala, G. Havur, S. Sperl, S. Steyskal, A. Haselböck, J. Mendling, A. Polleres, Shapeworks: A bpms extension for complex process management., in: BPM Demo Track, volume 1789 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 50–55.

[11] M. Gebser, M. Maratea, F. Ricca, The seventh Answer Set Programming Competition: Design and results, Theory and Practice of Logic Programming 20 (2020) 176–204.