Automated Abstract Planning with Use of Genetic Algorithms

Jaroslaw Skaruz
ICS, University of Natural Sciences and Humanities
3 Maja 54
08-110 Siedlce, Poland
jaroslaw.skaruz@uph.edu.pl

Artur Niewiadomski
ICS, University of Natural Sciences and Humanities
3 Maja 54
08-110 Siedlce, Poland
artur@ii.uph.edu.pl

Wojciech Penczek
Institute of Computer Science
Polish Academy of Sciences
Jana Kazimierza 5
01-248 Warsaw, Poland
penczek@ipipan.waw.pl

ABSTRACT

The paper presents a new approach based on nature inspired algorithms to an abstract planning problem, which is a part of the web service composition problem. An abstract plan is defined as an equivalence class of sequences of the same service types that satisfy a user query. The objective of our genetic algorithm (GA) is to return representatives of abstract plans without generating all the equivalent sequences.

Categories and Subject Descriptors

I.2.8. [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE

Keywords

genetic algorithm, web service composition

1. INTRODUCTION

The problem of finding a composition of web services that meets user goals is hard and well known as the Web Service Composition Problem (WSCP). There is a number of various approaches to solve WSCP [7, 1, 6]. In this paper we follow the approach of the system PlanICS [3] and focus on the first stage of WSCP, namely the Abstract Planning Phase (APP). APP makes use of *service types* and *object types* hierarchy defined in the *ontology*. A service type stands for a set of real-world services of similar capabilities, while the objects represent the data processed. A function assigning values to the object attributes is called a *valuation*. By a *world* we mean a pair consisting of a set of objects and a valuation function. The size of the set of the objects of a world w is denoted by $|w|$.

One of the crucial concepts of PlanICS consists in a world transformation. A service type transforms an *input* world into an *output* world by processing the set of input objects, possibly changing their states, and producing the output objects, according to the service type specification. *Transformation sequences* are sequences of service types, where each subsequent service type is able to transforma the world obtained from the previous transformation. A user query defines *initial* worlds to start with and *expected* worlds to be obtained as a result of the composition. A transformation sequence able to transform an initial into an expected world is called a *solution*. An *abstract plan* is defined as the set of solutions built over the same multiset of service types.

The main goal of APP is to reduce the search space for the next composition stage (i.e., concrete planning) by eliminating the services of the types not used in the abstract plans, and to provide a number of potential ways to realize the user query. This paper proposes a novel approach to APP based on GA and the multisets of service types as a new compact representation of the abstract plans. As far as the related work is concerned. The constrained optimization problem was considered in many papers (see [2]). A penalty function was applied in [4], but without any penalty parameter. A multiset representation of an individual was used in [8].

2. GA-BASED ABSTRACT PLANNING

Since the objective of GA is to find abstract plans, an individual is defined as a multiset of service types. Then, one has to check whether a multiset M represents an abstract plan. To this aim, a sequence of service types seq_M, for M, is constructed as follows. In the successive iterations we remove from M a service type s, which is able to transform[1] a current world w (starting from an initial world of the user query), and we append s to the sequence initialized to ϵ. Finally, we obtain a triple (seq_M, l_M, w_M), where seq_M is a transformation sequence, l_M is the sequence length, and w_M is the final world obtained by a transformation of an initial world by l_M subsequent service types of seq_M.

Fitness function.

The fitness function, which takes a triple (seq_M, l_M, w_M) and an expected world w_q as arguments, is calculated as:

$$fitness_M = \frac{f_{w_M} * \alpha + c_{w_M} * \beta + l_M * \gamma + g_{seq_M} * \delta}{|w_q| * \alpha + |w_q| * \beta + |M| * \gamma + |M| * \delta} \quad (1)$$

where: f_{w_M} is the maximal number of objects from w_M, which types and valuations are consistent with objects from w_q, $c_{w_M} = min\big(cst(w_M), |w_q|\big)$, where $cst(w_M)$ is the number of the objects from w_M, which types are consistent with objects from w_q, g_{seq_M} is the number of the *good*[2] *service types* occurring in seq_M, $\alpha = 0.1$, $\beta = 0.7$, $\gamma = 0.1$, and $\delta = 0.2$ are parameters of the fitness function.

[1] If there exists more then one such a service type, then one of them is chosen randomly.

[2] A good service type produces objects of types present in expected worlds or present in input worlds of other good services.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

After the first abstract plan is found, GA keeps searching for another solution. Let Sol denote a non empty family of multisets representing the abstract plans found. Then, the measure of similarity of a multiset M is computed as follows:

$$sim_M^{Sol} = max\left(\left\{ \frac{|M \cap S|}{|M|} \;\middle|\; S \in Sol \right\}\right) \quad (2)$$

Finally, when there are solutions found, the fitness value of the individual M is calculated according to Eq. 3:

$$fitness_M^{Sol} = fitness_M * (1.0 - sim_M^{Sol}) \quad (3)$$

Mutation operator.

One of our main contributions is a mutation operator specialized for the discussed problem, which takes advantage of the *good service type* concept. Thus, a gene is mutated only if it does not represent a good service type, and if there is a *good service type* for the sequence considered. To this aim, a set of all good service types is computed and its randomly selected element replaces the mutated gene.

3. EXPERIMENTAL RESULTS

We evaluated our algorithm using ontologies, user queries, and abstract plans provided by our ontology generator. The benchmarks were scaled according to several parameters: the number of service types in the ontology (from 64 to 256), the plan length (between 6 and 15), and the number of abstract plans (from 1 to 10). GA was run for 50 iterations, with population of 1000 individuals, with crossover and mutation probability equal to 95% and 5%, resp.

The overall results are quite promising. GA was able to find the solution with probability 100% for benchmarks with only one plan in the search space of size up to 2^{84}, consuming max. 22 sec. of total run time. Clearly, the probability of finding a solution decreases for larger search spaces, however we are still able to find it even in a search space of size 2^{120}, but then the GA runtime increases to 49 sec.

Fig. 1 presents the results of searching for many abstract plans consisting of 6 service types each. The first solution

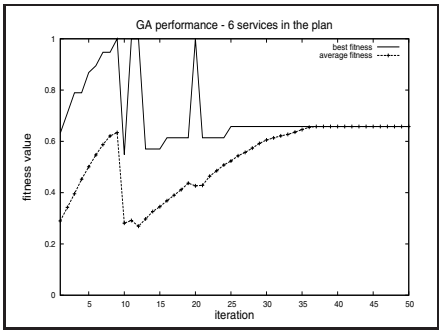

Figure 1: GA performance - 6 services in the plan

was found in the 9th iteration of GA and then, according to the similarity measure (3), the fitness of the best individual drastically decreased. In this case 4 of 10 different abstract plans have been found, however our GA was able to find up to 6 plans in a single run, consuming about 7.5 sec.

Fig. 2 displays the results of another experiment, where the search space of size 2^{105} contains only one abstract plan

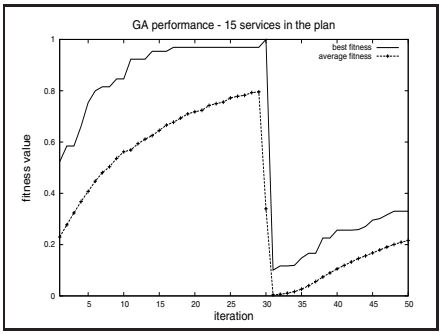

Figure 2: GA performance - 15 services in the plan

of length 15. The solution was found in the 30th iteration of GA, while the total runtime is equal to 35 sec.

4. CONCLUSIONS

A high efficiency of GA has been obtained using a special form of the fitness function and the dedicated mutation operator. To overcome the problem of finding very similar solutions, we have used multisets for representing abstract plans. Moreover, our algorithm is able to find many abstract plans in a single run. For more details see our Report [5].

Acknowledgments

This work has been supported by the Polish National Center for Science under the grant No. 2011/01/B/ST6/01477.

5. REFERENCES

[1] M. Blake, T. Weise, and S. Bleul. Wsc-2010: Web services composition and evaluation. In *Proc. of SOCA*, pages 1–4, 2010.

[2] C. A. C. Coello. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, 17:319–346, 2000.

[3] D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Polrola, and J. Skaruz. Harmonics - a tool for composing medical services. In *Proc. of ZEUS 2012*, pages 25–33, 2012.

[4] D. Kalyanmoy. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186:311–338, 2000.

[5] A. Niewiadomski, W. Penczek, and J. Skaruz. Towards automated abstract planning based on a genetic algorithm. Technical Report 1026, ICS PAS, 2012. http://artur.ii.uph.edu.pl/papers/rep1026.pdf.

[6] J. Peer. A pop-based replanning agent for automatic web service composition. In *Proc. of Conf. on the Semantic Web: Research and Application*, volume 3532 of *LNCS*, pages 189–198, 2005.

[7] J. Rao and X. Su. A survey of automated web service composition methods. In *Proc. of the Int. Workshop on Semantic Web Services and Web Process Composition*, volume 3387 of *LNCS*, pages 43–54, 2004.

[8] A. Wu and I. Garibay. The proportional genetic algorithm: Gene expression in a genetic algorithm. *Genetic Programming and Evolvable Machines*, 3(2):157–192, 2002.