Hybrid Approach to Abstract Planning of Web Services

Artur Niewiadomski
Institute of Computer Science
Siedlce UPH
Siedlce, Poland
e-mail: artur.niewiadomski@uph.edu.pl

Wojciech Penczek
Institute of Computer Science
Polish Academy of Science and UPH
Warsaw, Poland
e-mail: penczek@ipipan.waw.pl

Jaroslaw Skaruz
Institute of Computer Science
Siedlce UPH
Siedlce, Poland
e-mail: jaroslaw.skaruz@uph.edu.pl

***Abstract*—The paper deals with the abstract planning problem – the first stage of the Web Service Composition in the PlanICS framework. Abstract planning consists in finding (multisets of) service types which can potentially satisfy the user query. We introduce a novel planning technique based on a combination of Genetic Algorithm with a Satisfiability Modulo Theories Solver, which allows to improve the efficiency of each separate method. The paper presents also some experimental results which show the advantages of the hybrid method when applied to large search spaces with many alternative solutions.**

Keywords*-*Web Service Composition; Abstract Planning; Genetic Algorithm; Satisfiability Modulo Theories; Hybrid Algorithm

I. Introduction

The Web Service Composition (WSC) problem [1][2][3] consists in composing simple functionalities, accessible via well-defined interfaces, in order to achieve more complex objectives. This is one of the main ideas of Service-Oriented Architecture (SOA) [1]. Unfortunately, WSC is a complex and hard problem and therefore requires an application of quite sophisticated methods and tools.

PlanICS [4] is a framework aimed at WSC, easily adapting existing real-world services. The main assumption in PlanICS is that all the Web services in the domain of interest as well as the objects that are processed by the services, can be strictly classified in a hierarchy of *classes*, organised in an *ontology*. Another key idea is to divide the planning into several stages. The first phase deals with *classes of services*, where each class represents a set of real-world services, while the other phases work in the space of *concrete services*. The first stage produces an *abstract plan* composed of service classes [5]. Then, the Offer Collector (OC), i.e., a module of PlanICS, interacts with instances of the service types constituting the abstract plan and retrieves data used in the concrete planning (CP) phase. As a result of CP, a *concrete plan*, i.e., a sequence of offers satisfying the predefined optimization criteria is obtained. Such a multiphase approach enables to reduce dramatically the number of Web services to be considered and inquired for offers.

This paper deals with the Abstract Planning Problem (APP), which is known to be NP-hard [5]. Our previous works employed several heuristic methods to solve APP: Genetic Algorithm (GA) [6][7], a translation to Petri nets [8], and Satisfiability Modulo Theories (SMT) Solvers [5]. The results of the extensive experiments show that the proposed methods are complementary, but every single one suffers from some disadvantages.

The main disadvantage of an SMT-based solution is often a long computation time, which is not acceptable in the case of a real-world interactive planning tool. The translation to Petri nets seems to be an efficient planning method, but only for some specific types of ontologies. On the other hand, a GA-based approach is relatively fast, but the probability of finding a solution, as well as the number of solutions found, decrease with the increase of the plan lengths.

Thus, our aim consists in exploiting the advantages of the two abstract planning methods – based on GA and SMT – by combining them into one hybrid algorithm. The main idea of our hybrid approach involves a modification of the standard GA in such a way that after every iteration of GA several individuals are processed by the SMT-based procedure, which aims at modifying them in order to obtain solutions of APP.

In our previous papers [9][10], we showed several variants of hybrid algorithms for solving the Concrete Planning Problem (CPP). However, in the case of CPP we dealt with the constrained optimisation problem. The main goal was to find a concrete plan satisfying all the constraints and maximizing the quality function. Here, in case of APP, the main aim is to find *all* abstract plans with a fixed number of service types. In practice, this means finding as many alternative plans as possible, using available resources (e.g., computer memory and computation time). Therefore, the main contribution of this paper is a new version of a hybrid algorithm combining GA with SMT, which finds abstract plans. Since we build upon our previous work, the general idea is somehow similar to the one applied in [9]. However, due to the fundamental differences between CPP and APP, the realisation of the hybrid abstract planner is substantially different than the hybrid concrete ones. The details are discussed at the end of the next subsection.

A. Related Work

The existing solutions to the WSC problem are divided into several groups. Following [11] our hybrid algorithm belongs to the AI planning methods. Other approaches include: automata theory [12], situation calculus [13], Petri nets [14], theorem proving [15], and model checking [16]. In what follows, we shortly review the literature on composition methods related to our non-hybrid and hybrid algorithms.

A composition method closest to our SMT-based method [5] is presented in [17], where the authors reduce WSCP to a reachability problem of a state-transition system. The problem is encoded as a propositional formula and tested for satisfiability using a SAT-solver. This approach makes use of ontologies describing a hierarchy of types and deals with an inheritance relation. However, we consider also the states of the objects, while [17] deals with their types only. Moreover, among other differences, we use a multiset-based SMT encoding instead of Propositional Satisfiability Problem (SAT).

As far as our GA-based solution of APP is concerned, the approach closest to ours is given in [18], where GA is used to one phase planning, which combines an abstract and a concrete one. The idea of a multiset representation of a GA individual has been also used in [19]. However, contrary to our approach, no linearization of a multiset is generated in order to compute the fitness value.

Hybrid algorithms for WSC are also known in the literature. Jang et al. [20] use a combination of Ant Colony Algorithm with GA to find a concrete plan. While the experimental results are better than these obtained using a simple GA, the number of service types and offers used are not sufficient to draw general conclusions about efficiency of this approach. A combination [21] of two evolutionary algorithms, Tabu Search and GA, was also used to CP. Although this method allowed to find good quality concrete plans, our hybrid algorithm allows for dealing with much larger search spaces.

In our previous papers [9][10], we presented several variants of hybrid algorithms combining GA and SMT. One of them exploits an SMT-solver in order to (partially) generate the initial populations for GA. The other versions of hybrid algorithms are sightly modified GAs using an SMT-solver as a local search engine. That is, given an individual with a relatively high fitness value, an SMT-based procedure tries to change values of several genes in order to improve the individual. This approach is the closest to the method proposed in the present paper. Here, we also exploit an SMT-solver to improve some individuals of a population maintained by GA, but that is where the similarities end. The main differences result from various domains and the definitions of the abstract and concrete planning. Another difference is in what "the improvement" means in each case. In the case of CPP one is searching for such values of genes that satisfy the (predefined) constraints. However, satisfiability of the constraints is just one of the fitness function components, so such an improved individual is usually helpful for GA, but it is unlikely a final solution. On the other hand, in the case of the hybrid method applied to solve APP, if the SMT-based procedure returns an improved individual, then it already represents an abstract plan. The next group of differences are the technical details of the algorithms, like, e.g., the strategies regarding when, how often, and how many times SMT-solver should be run, which and how many genes are to be changed, and how to choose the individuals being good candidates to an improvement. Moreover, according to our best knowledge, there are no other approaches combining symbolic and evolutionary methods into hybrid algorithms.

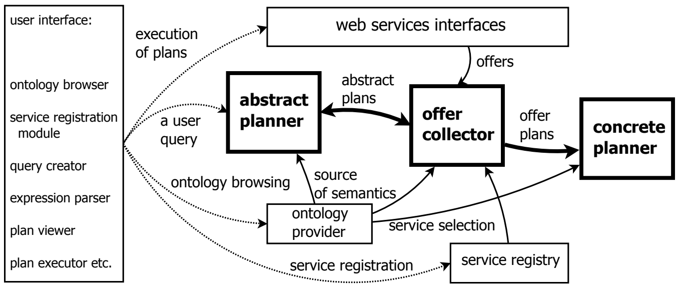

Figure 1. A simplified diagram of the PlanICS system architecture

The rest of the paper is structured as follows. In Section II, the PlanICS framework is introduced and APP is formulated. Section III presents the main ideas of our hybrid approach as well as some technical solutions. Next, the preliminary experimental results are given and discussed. The paper ends with some conclusions.

II. PlanICS Framework

This section sketches the main ideas behind the PlanICS framework and gives all the intuitions necessary to formulate the abstract planning problem. The formal definitions can be found in [5].

An ontology contains a system of *classes* describing the types of the services as well as the types of the objects they process. A class consists of a unique name and a set of the attributes. By an *object*, we mean an instance of a class. By a *state* of an object, we mean a valuation of its attributes. A set of objects in a certain state is called a *world*.

The main goal of the system is to find a composition of services that satisfies a user *query*. The query interpretation is defined by two sets of worlds: the initial and the expected one. Moreover, the query may include several additional constraint sets used at different planning stages. Figure 1 shows the general PlanICS architecture. The bold arrows correspond to computation of a plan while the thin arrows model the planner infrastructure. While this paper concerns APP, in what follows we focus on the abstract planning phase.

A. Abstract Planning Problem

The first stage of the composition in the PlanICS framework is the abstract planning. It consists in matching services at the level of input/output types and the abstract values. That is, because at this stage it is sufficient to know if an attribute does have a value, or it does not, we abstract from the concrete values of the object attributes, and use two special values **set** and **null**.

Thus, for a given ontology and a user query, the goal of the abstract planning is to find such a (multi)set of service types that allows to build a sequence of service types transforming an initial world of the user query into some *final world*, which has to be consistent with an expected world, also defined as a part of the query. The consistency between a final world and an expected one is expressed using the notion of the *compatibility* relation, formally defined in [5]. Intuitively, a final world W_f is compatible with an expected world W_e if the following conditions are met: (i) for every object $o_e \in W_e$ there exists a unique object $o_f \in W_f$, such that both the objects are of the same type or the type of o_f is a subtype of o_e; (ii) both the objects agree on the (abstract) values of the common attributes.

The result of the abstract planning stage is called a Context Abstract Plan (CAP). It consists of a multiset of service types (defined by a representative transformation sequence), contexts (mappings between services and the objects being processed), and a set of final worlds. However, our aim is to find not only a single plan, but many (significantly different, and all if possible) abstract plans, in order to provide a number of alternative ways to satisfy the query. We distinguish between abstract plans built over different multisets of service types. See [5] for more details.

III. Hybrid Approach to Solve APP

In this section, we recall some details of our GA-based abstract planner, as a base of the new hybrid algorithm. Then, an analysis of our previous experimental results obtained using GA and SMT is provided. Finally, basing on this analysis, the hybrid algorithm is proposed.

A. *Application of GA to solving APP*

The base of our hybrid approach is the standard GA aimed at solving APP. GA is a non deterministic algorithm maintaining a population of potential solutions during an evolutionary process. A potential solution is encoded in a form of a GA individual, which, in case of APP, is a sequence of natural values representing service types. However, since each abstract plan makes use of the multiset concept, the individuals differing only in the genes ordering are indistinguishable. In each iteration of GA, a set of individuals is selected for applications of genetic operations, such as the standard one-point crossover and mutation, which leads to obtaining a new population passed to the next iteration of GA. The selection of an individual and thus the promotion of its offspring to the next generation depends on the value of the fitness function.

Before the fitness function is calculated, every individual is processed by a procedure which is trying to find such an ordering of genes that an individual starts with a transformation sequence of a maximal length (see [6] for details). Moreover, such an executable prefix of an individual consists of *good service* types. Intuitively, a service type is *good* if it produces or modifies objects that are a part of some expected world, or they are an input of another good service type. Note that all the abstract plans are built solely of good service types only.

More details on applying GA to abstract planning can be found in [6][7].

B. *Analysis*

Let us begin with an analysis of the GA behaviour. Our previous works on an application of GA to solve APP show that a short computation time is certainly its advantage, but on the negative side the probability of finding a solution decreases along with the length of an abstract plan. Moreover, GA is unable to find solutions for instances with more than 6 services and several abstract plans in a search space.

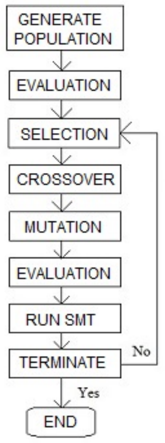

Figure 2. Hybrid algorithm flowchart.

On the other hand, the computation time of the SMT-based algorithm applied to APP is longer (comparing to GA), but all the solutions are found for instances with up to 9 services. Moreover, SMT performs a (symbolic) exploration of the whole search space to make sure that there is no other plan up to the given length. However, for instances with 12 services the SMT-solver is able to finish the computation within the given time only for search spaces containing one plan. For the remaining cases it indeed returns at least one solution, but it does not guarantee that more plans do not exist. The above discussion together with our previous successful applications of the hybrid algorithms to CPP [9][10], lead to a conclusion that a promising approach should consist in a combination of both approaches exploiting their advantages. To this aim we have to identify (and encode as an SMT formula) such a sub-problem of APP, which is solvable for an SMT-solver in a reasonable short time.

C. *Algorithm*

The problem we have encountered while solving some hard APP instances is as follows. Quite frequently GA ends up with unfeasible solutions containing only a few 'unmatched' services. This leads to the main idea of our hybrid approach. It relies upon an application of SMT to improve some number of the best individuals in each iteration of GA. An improvement consists in finding such good service types that can replace the unmatched ones. If such service types exist, then the modified individual represents an abstract plan. Figure 2 shows the consecutive steps of the hybrid algorithm.

Our hybrid algorithm mimics the standard steps of GA, which are modified in the following way. In each iteration of GA, after an evaluation step, a fixed number of top individuals (which do not yet constitute a complete abstract plan) are considered as candidates for an SMT-based improvement. Let I be an individual consisting of k genes. Let $e(I)$ denote the length of the maximal executable prefix of I and $g(I)$ denote the number of good service types of I. Then, the individual I is passed to the SMT-based procedure if the following conditions are met:

- $\lceil \frac{k}{2} \rceil \leq e(I) < k$, and

- $g(I) \geq \lceil \frac{k}{2} \rceil$.

Thus, only the individuals consisting at least in a half of good service types are considered as candidates for an improvement. Moreover, such a candidate should contain an executable prefix consisting of at least $\lceil \frac{k}{2} \rceil$ service types, but also having at least one gene to be changed.

If an optimal solution is found by SMT, then it is passed back to the GA population. The fixed number of individuals passed to the SMT procedure assures that only the individuals that have a real chance to be improved are considered for passing to SMT. Otherwise, this could result in a too long computation time of the SMT-based procedure. While an optimal solution is unknown a priori, it is very difficult to depict whether an individual is near to an abstract plan in the search space and thus if it is suitable for an improvement. In our approach this assessment is based on the two features of an individual: the length of its maximal executable prefix and the number of its good service types. Note that in the case of an abstract plan both the values are equal to k.

D. SMT-based problem encoding

The SMT procedure combined with GA is based on the encoding exploited in our "pure" SMT-based concrete planner [5]. However, now the task for an SMT-solver is to check whether the maximal (non-executable) suffix of an individual can be modified to make an improved individual a solution of APP. Thus, in addition to the user query q and the ontology, the SMT-based procedure takes also as input an individual and the length of its maximal executable prefix. The general idea of the SMT task is depicted in Figure 3.

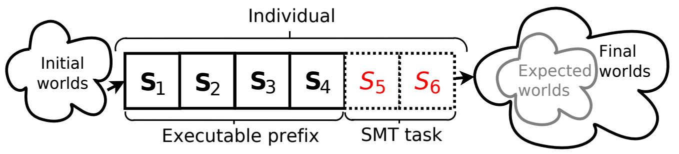

Figure 3. SMT role in the hybrid algorithm

Let I denote an individual in the form of a sequence of service types and I_j be the service type at the j-th position of the sequence. Let e denote the length of the maximal executable prefix and let k be the total length of I. Then, the task of an SMT-solver is reduced to checking satisfiability of the following formula:

$$\varphi_k^q = \mathcal{I}^q \bigwedge_{j=1..e} \Big(\mathcal{C}_j \wedge (s = I_j) \wedge \mathcal{T}_j^s\Big) \bigwedge_{i=(e+1)..k} \Big(\mathcal{C}_i \bigvee_{s \in \mathbb{S}} \mathcal{T}_i^s\Big) \wedge \mathcal{E}_k^q \wedge \mathcal{B}_k^q \tag{1}$$

where $\mathcal{I}^q$ and $\mathcal{E}_k^q$ are the formulas encoding the initial and the expected world of the user query, respectively, $\mathcal{C}_i$ encodes the i-th context function, $\mathbb{S}$ is the set of all service types from the ontology, and $\mathcal{T}_i^s$ encodes the worlds transformation by a service type s. $\mathcal{B}_k^q$ stands for a formula preventing from finding the already known solutions, if there are any. See [5] for more encoding details. Thus, an SMT-solver tries to find a sequence of service types of length $(k-e)$, which can replace the non-executable suffix of I in such a way that the improved individual is a (previously unknown) solution of APP.

IV. Experimental Results

We have evaluated the efficiency of the hybrid algorithm against the GA and SMT-based planners using the ontologies and the user queries generated by our software - Ontology Generator (OG). All ontologies are generated by OC in a random manner meeting the semantic rules. Each query is also generated randomly in such a way that the number of various abstract plans equals to the value of a special parameter of OG. This guarantees that we know a priori whether the planners have found all solutions.

A. Configuration of the Experiments

In order to evaluate the efficiency of our planners, we have conducted experiments using twenty four instances of APP generated by OG, with the following parameter values: the solution length (the number of service types constituting a plan): $k \in \{6, 9, 12, 15\}$, the number of the service types in the ontology: $n \in \{64, 128, 256\}$, and the number of the plans in the search space: $sol \in \{1, 10\}$. Thus, the size of the search space varies from $64^6 = 2^{36}$ for the first and the fourth experiment, up to $256^{15} = 2^{120}$ in the case of the 21st and the 24th instance.

Each experiment has been repeated 20 times on a standard PC with 2GHz CPU, 8GB RAM, and Z3 [22] version 4.3 as an SMT-solving engine. The experiments involving the "pure" GA and hybrid planner have been performed using the following parameters: the population size equals 100 for the instances with one solution only and 500 for the instances with ten plans in the search space, the number of the iterations equals 100, whereas the probabilities of crossover and mutation are at levels of 95% and 0.5%, respectively. We impose the 2000 sec. time limit for every experiment.

B. Experimental Results Evaluation

A comparison of the experimental results of all the three planners is presented in Table I. The best results are marked with bold. The first four columns from the left contain the experiment number and the OG parameters used to generate the instances. The next six columns present the results of our hybrid planner, such as the time consumed by the SMT-based procedure calls and consumed by GA, the total computation time, the average and the maximal number of plans found, as well as the probability of finding a plan. Note that the average number of plans found is computed taking into account only these cases, where at least one solution has been found. The next four columns contain the results obtained using the GA-based abstract planner. That is, from left to right: the computation time, the average and the maximal number of plans found, and the probability of finding a plan. Finally, the last two columns display: the time consumed by the SMT-based abstract planner in order to find the first plan and the total computation time. Since the SMT-based planner always finds all the plans (provided it has enough time and memory), we do not report any other results here. Note that the memory usage does not exceed 2GB, even during the experiments with the largest instances and the "pure SMT" as the planning method. The SMT-based planner seems to be the most memory-demanding in the last phase of searching, i.e., while checking that there is no more plans and (symbolically)

TABLE I. Experimental results

				Hybrid						Pure GA				Pure SMT	
Exp	**k**	**n**	**sol**	**SMT** [s]	**GA** [s]	**Total** **[s]**	**Avg** plans	**Max** plans	**Prob.** [%]	**Time** [s]	**Avg** plans	**Max** plans	**Prob.** [%]	**First** [s]	**Total** [s]
1	6	64	1	4.05	8.24	12.29	1	1	100	**5.71**	**1**	**1**	**100**	6.31	12.8
2		128		5.77	8.78	14.55	1	1	100	**8.07**	**1**	**1**	**100**	7.29	14.8
3		256		10.79	13.29	24.07	1	1	100	**13.62**	**1**	**1**	**100**	16.66	27.1
4		64	10	3.04	25.74	28.78	3.25	10	100	24.29	5.4	10	100	**5.22**	**18.3**
5		128		6.52	32.15	38.67	3.15	8	100	31.21	6.25	10	100	**8.54**	**26.6**
6		256		13.85	43.33	57.18	3.65	8	100	45.95	5.55	9	100	**11.93**	**38.1**
7	9	64	1	12.08	11.67	23.75	1	1	85	**11.83**	**1**	**1**	**95**	19.49	58.7
8		128		25.65	15.68	41.33	1	1	90	**13.43**	**1**	**1**	**100**	41.01	90.1
9		256		43.61	28.88	72.49	1	1	90	**26.74**	**1**	**1**	**90**	54.99	133
10		64	10	**17.54**	**56.9**	**74.43**	**3.15**	**10**	**100**	57.69	1.77	4	65	21.09	295
11		128		**30.64**	**63.38**	**94.02**	**4.16**	**10**	**95**	69.94	1.54	4	65	49.93	553
12		256		**61.64**	**113.05**	**174.69**	**4.32**	**10**	**95**	113.15	1.33	2	30	113.3	977
13	12	64	1	55.09	21.77	76.86	1	1	45	**21.22**	**1**	**1**	**65**	156.4	781
14		128		86.48	30.15	116.62	1	1	85	**28.12**	**1**	**1**	**60**	203.2	1962
15		256		118.7	46.82	165.52	1	1	55	**46.31**	**1**	**1**	**60**	315.4	1947
16		64	10	**78.98**	**118.56**	**197.54**	**2.79**	**10**	**95**	118.29	0	0	0	113.5	> 2000
17		128		**109.89**	**139.96**	**249.84**	**2.38**	**10**	**80**	148.65				250.5	
18		256		**193.17**	**253.22**	**446.39**	**1.85**	**6**	**65**	260.94				325.8	
19	15	64	1	119.09	33.68	152.77	1	1	25	**34.56**	**1**	**1**	**30**	469.7	
20		128		185.34	43.17	228.51	1	1	30	**40.45**	**1**	**1**	**25**	382.1	
21		256		247.3	68.26	315.56	1	1	35	**68.69**	**1**	**1**	**35**	1018	
22		64	10	**168.46**	**237.57**	**406.03**	**1.67**	**3**	**30**	216.6	0	0	0	413	
23		128		**309.53**	**267.83**	**577.36**	**3**	**5**	**10**	261.21				1850	
24		256		**304.88**	**450.63**	**755.5**	**3**	**3**	**5**	437.59				931	

exploring the whole search space. The memory usage of the "pure" GA algorithm remains below several hundred MB, and in the case of the hybrid method it does not exceed 1GB for the largest instances.

It is easy to observe that in the case of the instances with only one plan in the search space the "pure" GA algorithm is superior to the other algorithms due to its relative low computation time. However, it is worth noticing that the probability of finding a solution by GA drops rapidly with the increase of the lengths of plans. On the other hand, analysing the experiments where there are ten plans in the search space, one can notice that the "pure" SMT algorithm is faster than the other algorithms only for APP instances having the shortest solutions. In all the other cases the hybrid planner is clearly superior. Concerning the experiments 10, 11, and 12, the computation time of the hybrid planner is much lower than the time consumed by the "pure" SMT-based algorithm. Comparing the results obtained using the hybrid and the "pure" GA algorithm it is easy to see that admittedly GA is about 30% faster, but the hybrid one outperforms GA if all other measures are considered.

The analysis of the remaining experiments, where the search spaces contain ten solutions is even simpler. These are the most important results related to the instances which turned out be hard for the pure GA and SMT planner, as GA does not yield any result while the SMT-based planner is running out of time. The hybrid algorithm outperforms the other planners for the instances, for which the plans consist of more than 6 service types. In case of 12 services, the maximal number of the abstract plans found equals to 10, while for 15 services it is 5. Summarizing, the experiments 16, 17, 18, 22, 23, and 24 prove that applying the hybrid algorithm to APP one can obtain substantially better results.

C. Comparison with Other Approaches

We have done our best to compare efficiency of our tool with another system. Nam et al. [17] report 7 experiments performed on a set of 413 *concrete* Web services, where SAT-time consumed for every composition varies from 40 to 60 sec. However, the composition consists only in simple type matching, the plans consist of a few services only, and the main goal is to find the shortest sequence of services satisfying the user query. We have repeated these experiments translating first the input data to the PlanICS ontology. We treated each concrete service as a service type, and we modelled the service parameters type hierarchy as the object types. Our results have appeared to be better. PlanICS is able to find the shortest solution in just fractions of a second of SAT-time and in several seconds of the total computation time. Overall, our approach is more complex and the composition by just types matching is a special, simplified case of our planning. Moreover, instead of searching just for the shortest solution, we focus on finding a number of alternative plans.

For the second comparison we have used the Fast Downward tool [23] that is aimed at solving problems specified in PDDL (Planning Domain Definition Language) [24]. We developed a tool which translates PlanICS ontologies and user queries into PDDL domains and problems, respectively. Then, we translated the benchmarks described in Sec. IV and we used them as input for FD. The FD tool was able to find a solution only for smallest instances with 64 service types. For these cases an interesting observation can be made. Namely, the amount of time and memory consumed by FD is not so strongly related to the length of the plan, like in the case of PlanICS, but they clearly depend from number of services in the ontology. For larger benchmarks the whole available memory has been quickly consumed and the computation had to be aborted. It shows the advantage of PlanICS while reasoning in large ontologies.

V. Conclusion and Future Work

A new hybrid algorithm based on GA and SMT has been proposed to solve APP. The algorithm has been implemented and some preliminary experiments have been performed for benchmarks having different sizes of ontologies and solutions in the search space. The very first results show that using a combination of the SMT- and GA-based approach, one can obtain quite good results, especially for problems that are hard to solve using the "pure" planning methods, i.e., with large search spaces and many alternative solutions.

We plan to further improve the efficiency of our hybrid approach in terms of lower computation times and higher probabilities of finding solutions. The successful application of the hybrid algorithm to abstract planning problem shows that the proposed method has a high potential. Thus, another important task to be addressed in a future work is an application of similar algorithms to other hard problems involving an exploration of huge search spaces.

ACKNOWLEDGMENT

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

REFERENCES

[1] M. Bell, Introduction to Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture. John Wiley & Sons, 2008, ISBN: 978-0-470-14111-3.

[2] S. Ambroszkiewicz, "Entish: A Language for Describing Data Processing in Open Distributed Systems," Fundam. Inform., vol. 60, no. 1-4, 2003, pp. 41–66.

[3] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in Proc. of SWSWPC'04, ser. LNCS, vol. 3387. Springer, 2005, pp. 43–54.

[4] D. Doliwa et al., "PlanICS - a Web Service Compositon Toolset," Fundam. Inform., vol. 112(1), 2011, pp. 47–71. [Online]. Available: http://dx.doi.org/10.3233/FI-2011-578

[5] A. Niewiadomski and W. Penczek, "Towards SMT-based Abstract Planning in PlanICS Ontology," in Proc. of KEOD 2013 International Conference on Knowledge Engineering and Ontology Development, September 2013, pp. 123–131.

[6] J. Skaruz, A. Niewiadomski, and W. Penczek, "Evolutionary Algorithms for Abstract Planning," in PPAM (1), ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds., vol. 8384. Springer, 2013, pp. 392–401.

[7] ——, "Solving the abstract planning problem using genetic algorithms," Studia Informatica, vol. 1-2(17), 2013, pp. 29–48, ISSN: 1731-2264.

[8] A. Niewiadomski and K. Wolf, "LoLA as Abstract Planning Engine of PlanICS," in Proceedings of the International Workshop on Petri Nets and Software Engineering, co-located with 35th International Conference on Application and Theory of Petri Nets and Concurrency (PetriNets 2014) and 14th International Conference on Application of Concurrency to System Design (ACSD 2014), Tunis, Tunisia, June 23-24, 2014, pp. 349–350. [Online]. Available: http://ceur-ws.org/Vol-1160/paper26.pdf

[9] A. Niewiadomski, W. Penczek, and J. Skaruz, "Genetic Algorithm to the Power of SMT: a Hybrid Approach to Web Service Composition Problem," in Service Computation 2014 : The Sixth International Conferences on Advanced Service Computing, 2014, pp. 44–48.

[10] ——, "A Hybrid Approach to Web Service Composition Problem in the PlanICS Framework," in Mobile Web Information Systems, ser. Lecture Notes in Computer Science, I. Awan, M. Younas, X. Franch, and C. Quer, Eds. Springer International Publishing, 2014, vol. 8640, pp. 17–28. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10359-4_2

[11] Z. Li, L. O'Brien, J. Keung, and X. Xu, "Effort-Oriented Classification Matrix of Web Service Composition," in Proc. of the Fifth International Conference on Internet and Web Applications and Services, 2010, pp. 357–362.

[12] S. Mitra, R. Kumar, and S. Basu, "Automated Choreographer Synthesis for Web Services Composition Using I/O Automata," in ICWS, 2007, pp. 364–371.

[13] V. Chifu, I. Salomie, and E. St. Chifu, "Fluent calculus-based Web service composition - From OWL-S to fluent calculus," in Proc. of the 4th Int. Conf. on Intelligent Computer Communication and Processing, 2008, pp. 161 –168.

[14] V. Gehlot and K. Edupuganti, "Use of Colored Petri Nets to Model, Analyze, and Evaluate Service Composition and Orchestration," in System Sciences, 2009. HICSS '09., jan. 2009, pp. 1 –8.

[15] J. Rao, P. Küngas, and M. Matskin, "Composition of semantic web services using linear logic theorem proving," Inf. Syst., vol. 31, no. 4, Jun. 2006, pp. 340–360.

[16] P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes," in The Semantic Web ISWC 2004, ser. LNCS, 2004, vol. 3298, pp. 380–394.

[17] W. Nam, H. Kil, and D. Lee, "Type-Aware Web Service Composition Using Boolean Satisfiability Solver," in Proc. of the CEC'08 and EEE'08, 2008, pp. 331–334.

[18] F. Lecue, M. D. Penta, R. Esposito, and M. Villani, "Optimizing QoS-Aware Semantic Web Service Composition." in Proceedings of the 8th International Semantic Web Conference, 2009, pp. 375–391.

[19] I. Garibay, A. S. Wu, and O. Garibay, "Emergence of genomic self-similarity in location independent representations," Genetic Programming and Evolvable Machines, vol. 7(1), 2006, pp. 55–80.

[20] Z. Jang, C. Shang, Q. Liu, and C. Zhao, "A Dynamic Web Services Composition Algorithm Based on the Combination of Ant Colony Algorithm and Genetic Algorithm," Journal of Computational Information Systems, vol. 6(8), 2010, pp. 2617–2622.

[21] J. A. Parejo, P. Fernandez, and A. R. Cortes, "QoS-Aware Services composition using Tabu Search and Hybrid Genetic Algorithms," Actas de los Talleres de las Jornadas de Ingenieria del Software y Bases de Datos, vol. 2(1), 2008, pp. 55–66.

[22] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in Proc. of TACAS'08, ser. LNCS, vol. 4963. Springer-Verlag, 2008, pp. 337–340.

[23] M. Helmert, "The Fast Downward Planning System," Journal of Artificial Intelligence Research, vol. 26, 2006, pp. 191–246.

[24] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners," Artificial Intelligence, vol. 173, no. 5, 2009, pp. 619–668.