Genetic Algorithm to the Power of SMT: a Hybrid Approach to Web Service Composition Problem

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

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

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

Abstract—The paper deals with the concrete planning problem – a stage of the Web Service Composition in the Planics framework, which consists in choosing the best service offers in order to satisfy the user query and to maximize the quality function. We introduce a novel planning technique based on a combination of a Genetic Algorithm with a Satisfiability Modulo Theories Solver, which allows to obtain better results than each of the methods separately. The paper presents some preliminary, although very encouraging, experimental results.

Keywords—Web Service Composition; Concrete Planning; Genetic Algorithm; Satisfiability Modulo Theories; Hybrid Algorithm

I. INTRODUCTION

One of the fundamental ideas of Service-Oriented Architecture (SOA) [1] is to compose simple functionalities, accessible via well-defined interfaces, in order to realize more sophisticated objectives. The problem of finding such a composition is hard and known as the Web Service Composition (WSC) problem [1][2][3].

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]. Next, offers are retrieved by the Offer Collector (OC), a module of Planics, and used in the concrete planning (CP). As a result of CP, a concrete plan is obtained, which is a sequence of offers satisfying predefined optimization criteria. Such an approach enables to reduce dramatically the number of web services to be considered, and inquired for offers.

This paper deals with the Concrete Planning Problem (CPP), shown to be NP-hard [6]. Our previous works employ several techniques to solve it: a Genetic Algorithm (GA) [7], numeric optimization methods [8] as well as Satisfiability Modulo Theories (SMT) Solvers [6]. 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. On the other hand, a GA-based approach is relatively fast, but it yields solutions, which are far from optimum and found with a low probability.

Thus, our aim is to exploit the advantages of both methods by combining them into one hybrid algorithm, which is the main contribution of this paper. The main idea of our new hybrid approach involves a modification of the standard GA, such that after every couple of iterations of GA, several top-ranked individuals are processed by the SMT-based algorithm in order to improve them.

Over the last few years, CPP has been extensively studied in the literature. G. Canfora et al. [9] use a simple GA to obtain a good quality concrete plan. Y. Wu et al. [10] transforms CPP to a multi-criteria optimization problem and exploits GA to find a concrete plan. However, the authors present the experiments on a relatively small search space that could not provide valuable conclusions.

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

II. CONCRETE PLANNING PROBLEM

This section introduces the main ideas behind the Planics framework and gives all the necessary definitions for defining the concrete planning problem.

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. A key notion of Planics is that of a service. We assume that each service processes a set of objects, possibly changing values of their attributes, and produces a set of new (additional) objects. We say that a service transforms a world. The types of services available for planning are defined as elements of the branch of
The responsibility of OC is to collect a number of offers, where every offer represents one possible execution of a single service. However, other important tasks of OC are: (1) building a set of constraints resulting from the user query and from semantic descriptions of service types, and (2) a conversion of the quality constraints expressed using objects from the user query to an objective function built over variables from offer sets. Thus, we can formulate CPP as a constrained optimization problem.

Definition 2 (CPP): Let \( n \) be the length of CAP and let \( \emptyset = (O^1, \ldots, O^n) \) be the vector of offer sets collected by OC such that for every \( i = 1, \ldots, n \)

\[
O^i = \begin{bmatrix}
  o^i_{1,1} & \cdots & o^i_{1,m_1} \\
  \vdots & \ddots & \vdots \\
  o^i_{k_i,1} & \cdots & o^i_{k_i,m_i}
\end{bmatrix},
\]

and the \( j \)-th row of \( O^i \) is denoted by \( P^i_j \). Let \( \mathbb{P} \) denote the set of all possible sequences \( \{P^1, \ldots, P^n\} \), such that \( j \in \{1, \ldots, k_i\} \) and \( i \in \{1, \ldots, n\} \). The Concrete Planning Problem is defined as:

\[
\max \{Q(S) \mid S \in \mathbb{P}\} \text{ subject to } \mathbb{C}(S),
\]

where \( Q : \mathbb{P} \to \mathbb{R} \) is an objective function defined as the sum of all quality constraints and \( \mathbb{C}(S) = \{C_j(S) \mid j = 1, \ldots, c\} \), for \( c \in \mathbb{N} \), where \( S \in \mathbb{P} \), is a set of constraints to be satisfied.

Finding a solution of CPP consists in selecting one offer from each offer set such that all constraints are satisfied and the value of the objective function is maximized. This is the goal of the third planning stage and the task of a concrete planner.

Example. Consider a simple ontology describing a fragment of some financial market consisting of service types inheriting from the class Investment, which represent various types of financial instruments. Moreover, the ontology contains three object types: Money having the attribute amount, Transaction having the two attributes amount and profit, and Charge having the attribute fee. Suppose that each investment service takes \( m \) - an instance of Money as input, produces \( t \) and \( c \) - instances of Transaction and Charge, and updates the amount of money remaining after the operation, i.e., the attribute \( m.amount \). Assume that the user would like to invest up to $100 in three financial instruments, but he wants to locate more than $50 in two investments. Moreover, the user wants to maximize the sum of profits and wants to use only services of handling fees less than $3. The latter two conditions can be expressed as an appropriate quality function and an aggregate condition. Consider an exemplary CAP consisting of three instances of the Investment service type. A single offer collected by OC is a vector \([v_1, v_2, v_3, v_4, v_5]\), where \( v_1 \) corresponds to \( m.amount \), \( v_2 \) to \( t.amount \), \( v_3 \) to \( t.profit \), and \( v_4 \) to \( c.fee \). Since the attribute \( m.amount \) is updated during the transformation, the offers should contain values from the world before and after the transformation. Thus \( v_5 \) stands for the value of \( m.amount \) after modification. Assuming that instances of Investment return \( k_1, k_2, \) and \( k_3 \) offers in response to the subsequent inquiries, we obtain three offer sets: \( O^1, O^2, \) and \( O^3 \), where \( O^i \) is a \( k_i \times 5 \)
matrix of offer values. The conditions from the query are translated to the following constraints: $C_1 := (a_{1,1}^{1} \leq 100)$ and $C_2 := (a_{1,2}^{1} + a_{2,2}^{2} > 50)$, where $i_1$, $i_2$, and $i_3$ are variables ranging over $1 \ldots k_i$. Moreover, the amount of money left after the operation is an input for the next investment. Thus, we have: $C_3 := (o_{1,5}^{1} = o_{2,1}^{2})$ and $C_4 := (o_{2,5}^{2} = o_{3,1}^{3})$. The aggregate condition is translated to the following constraint: $C_5 := (max\{o_{1,4}^{1}, o_{2,4}^{2}, o_{3,4}^{3}\} < 3)$, while the quality expression is translated to the quality constraint $Q_1 := \sum_{j=1}^{3} o_{ij}^{j,3}$.

III. A HYBRID SOLUTION AND PRELIMINARY RESULTS

The analysis of several hard CPP instances is our main motivation to combine the power of SMT with the potential of GA. The main disadvantage of a “pure” SMT-based solution is often a long computation time, which is not acceptable in the case of a real-world interactive planning tool. On the other hand, a GA-based approach is relatively fast, but it yields solutions, which are far from optimum and found with low probability. Thus, our aim is to exploit the advantages of both the methods by combining them into one hybrid algorithm.

A. Overview

The main idea is as follows. The base of our hybrid approach is the standard GA aimed at solving CPP. 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 CPP, is a sequence of natural values. 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. The fitness value of an individual is the sum of the optimization objective and the ratio of the number of the satisfied constraints to the number of all the constraints (see Def. 2), multiplied by some constant $\beta$:

$$fitness(I) = q(S_I) + \beta \cdot \frac{|sat(C(S_I))|}{c},$$  

(2)

where $I$ stands for an individual, $S_I$ is a sequence of the offer values corresponding to $I$, $sat(C(S_I))$ is a set of the constraints satisfied by a candidate solution represented by $I$, and $c$ is the number of all constraints. The role of $\beta$ is to reduce both the components of the sum to the same order of magnitude and to control the impact of the components on the final result. The value of $\beta$ depends on the estimation of the minimal and the maximal quality function value.

The main idea of our new hybrid approach involves the following modification of the standard GA. After every couple of iterations of GA, several top-ranked individuals are processed by the SMT-based algorithm. Given an individual $I$, the procedure searches for a similar, but improved individual $I'$, which represents a solution satisfying all the constraints and having a greater value of the objective function at the same time. The similarity between $I$ and $I'$ consists in sharing a number of genes. We refer to the problem of finding such an individual as to the Search for an Improved Individual (SFII).

Since there are many possible ways to exploit this idea, we start from the one which randomly selects the genes to be changed. The overview of our hybrid algorithm is depicted in Figure 2.

The SMT procedure combined with GA is based on the encoding exploited in our “pure” SMT-based concrete planner [6][8]. The idea is to encode SFII as an SMT formula which is satisfiable if such an individual exists. First, we initialize an SMT-solver allocating a set $V$ of all necessary variables:

- $oid^i$, where $i = 1 \ldots n$ and $n$ is the length of the abstract plan. These variables are needed to store the identifiers of offers constituting a solution. A single $oid^i$ variable takes a value between 1 and $k_i$.
- $o^i_j$, where $i = 1 \ldots n$, $j = 1 \ldots m_i$, and $m_i$ is the number of offer values in the $i$-th offer set. We use them to encode the values of $S$, i.e., the values from the offers chosen as a solution. From each offer set $O^i$ we extract the subset $R^i$ of offer values which are present in the constraint set and in the quality function, and we allocate only the variables relevant for the plan.

Next, using the variables from $V$, we encode the offer values, the objective function, and the constraints, as the formulas shared by all calls of our SMT-procedure. The offer values from the offer sets $O = (O^1, \ldots, O^n)$ are encoded as the formula

$$ofr(O, V) = \bigwedge_{i=1}^{n} \bigvee_{d=1}^{k_i} \left(oid^i = d \land \bigwedge_{o_{d,j} \in R^i} o^i_j = o^i_{d,j}\right).$$  

(3)

The formulae $ctr(C(S), V)$ and $qual(Q(S), V)$, denoted as $ctr$ and $q$ for short, encode the constraints and the objective function, respectively. Details are provided in [6].

Let $I = (g_1, \ldots, g_n)$ be an individual, $M = \{i_1, \ldots, i_k\}$ the set of indices of genes allowed to be changed, and $q(S_I)$ the value of the objective function where $n, k \in \mathbb{N}$.

Hence, the SFII problem is reduced to the problem of satisfiability of the following formula:

$$\bigwedge_{i \in \{1, \ldots, n\} \setminus M} (oid^i = g_i) \land ofr(O, V) \land ctr \land (q > q(S_I))$$  

(4)

That is, the formula (4) is satisfiable only if there exists an individual $I' = (g'_1, \ldots, g'_n)$ satisfying all the constraints, where $\forall g_{i \not\in M} g_i = g'_i$ and $q(S_I) > q(S_I')$, i.e., sharing with $I$ all genes of indices outside $M$ and having the larger value of objective function than $I$. If the formula is satisfiable, then
the values of the changed genes are decoded from the model returned by the SMT-solver, and the improved individual $I'$ replaces $I$ in the current population.

B. Experimental Results

As benchmarks for our experiments we choose four instances of CPP, which turned out to be difficult to solve using our “pure” SMT- and GA-based planner [6][8]. All the instances represent plans of length 15. Each offer set of Instances 1 and 3 contains 256 offers, hence, the number of the potential solutions equals $256^{15} = 2^{120}$. In the case of Instances 2 and 4 each offer set consists of 512 offers, thus the size of the search space is $512^{15} = 2^{135}$. The objective functions are as follows:

$$Q_{1.2} = \sum_{i=1}^{n} o_{j,i,1}, \quad Q_{3.4} = \sum_{i=1}^{n} (o_{j,i,1} + o_{j,i,2}),$$

(5)

while the set of the constraints is the same for all instances, and is defined as:

$$C = \{ (o_{i,j,2} < o_{j,i+1,2}) \}, \quad \text{for } i = 1, \ldots, n - 1.$$  

(6)

Besides the ordinary parameters of GA (which have been set to the same values as in pure GA), that is, the population size (1000), the number of iterations (100), the crossover and mutation probabilities (95% and 0.5% respectively), we introduce also parameters influencing the behaviour of the SMT component. Namely, when to start the SMT procedure for the first time (in the 20th iteration), how often the SMT procedure should be run (the parameter int stands for the number of the iterations between the subsequent SMT calls), the number of individuals passed to the SMT-solver during one iteration (parameter inds), and how many genes are allowed to be changed by SMT (the parameter ch.genes). Every instance has been tested 12 times, using a different combination of the parameters combination, and every experiment has been repeated 30 times on a standard PC with 2.8GHz CPU and Z3 [11] version 4.3 as SMT-solver engine.

The preliminary results of applying our new hybrid algorithm to Instances 1 and 2 are presented in Table I, where the columns from left to right display the parameter values and for each Instance, the total runtime of the algorithm ($t[s]$), the average quality of solutions found (avgQ), and the probability of finding a solution (P). For reference, we report in the two bottom rows the results of the pure SMT- and GA-based planners. One can easily see that quality values obtained in every experiment are higher than these returned by GA. However, in several cases the runtime or probability is not acceptable. We marked in bold the results, where the probability of finding a solution is at least 40% and the runtime is lower than that of the pure SMT-based planner.

For Instances 3 and 4, which objective function is more difficult, the results are given in Table II. Still, in some cases, the results are better than these returned by the pure planning methods. Note that the pure SMT-based algorithm was not able to find the optimal solution within given time limit (500 sec.).

Although the results are encouraging, the hybrid solution is clearly a trade-off between the three measures: the quality, the probability, and the computation time of the pure algorithms. In order to compare the results obtained taking all the measures into account at the same time, we define four simple score functions: $score_i = score(P, t, avgQ) = \sum_i (avgQ - const_i)$, where $P$, $t$, and $avgQ$ stand for the probability, the computation time, and the average quality, respectively, and $const_i$ is a parameter, which value is selected in such a way that for each Instance $i$ from I to IV, the score of the pure GA- and SMT-based algorithm is the same. These scores are the benchmarks for the comparison given in Figure 3. The values on the X-axes correspond to the rows of Table I and II, while the Y-axes indicate the values of the score functions. The black bars stand for the best hybrid results in comparison with the pure SMT- and GA-based algorithms. Notice that the hybrid algorithm can improve the solution score of each pure algorithm from 2 times (Instance 4) to nearly 6 times (Instance 1).

### IV. Conclusion and Future Work

The prototype of the hybrid concrete planner has been implemented and some preliminary experiments have been performed. The very first results show that even using a simple, or a naive strategy of combining the SMT- and GA-based approach, one can obtain surprisingly good results. We believe that the proposed method has a big potential. We plan to further improve the efficiency of our hybrid approach in terms of: a better quality of solutions, lower computation times, as well as higher probabilities of finding solutions. Another important task to be addressed in a future work is to investigate how to choose the parameter values, in order to get a trade-off between quality, probability, and the computation time desired by the user. Moreover, using the experience gained from the concrete
planning, we intend also to develop a hybrid solution for the abstract planning stage.

ACKNOWLEDGMENT

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

REFERENCES


