Combining Genetic Algorithm and SMT into Hybrid Approaches to Web Service Composition Problem

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

Wojciech Penczek
Institute of Computer Science
PAN Warsaw and Siedlce University, Poland
e-mail: wpenczek@gmail.com

Jaroslaw Skaruz
Institute of Computer Science
Siedlce University, 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 (GA) with a Satisfiability Modulo Theories (SMT) solver, which allows to obtain better results than each of the methods separately. We give three versions of a hybrid algorithm. Two of them involve a modification of the standard GA in such a way that after every couple of iterations of GA, several top-ranked individuals are processed by the SMT-based algorithm in order to improve them. The third one exploits an SMT-solver in order to generate the initial populations for GA, which results in a substantial improvement in the overall algorithm efficiency. The paper presents experimental results, which seem to be very encouraging.**

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

I. INTRODUCTION

This paper is an improved and extended version of the Service Computation 2014 conference paper "Genetic Algorithm to the Power of SMT: a Hybrid Approach to Web Service Composition Problem" [1]. This work introduces the two new versions of our hybrid algorithm, namely Semi-Random and InitPop Hybrid. The former is a slightly modified variant of the algorithm provided in the original paper, but due to the introduced changes it is also much more powerful. The latter implements our brand new concept of combining Satisfiability Modulo Theories and Genetic Algorithms. This is our main original contribution since the InitPop Hybrid algorithm is presented here for the very first time. In comparison with [1], this paper extends also the experimental results section and widely discusses the related work.

One of the fundamental ideas of Service-Oriented Architecture (SOA) [2] 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 [2][3][4].

PlanICS [5] 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 [6]. 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. Dividing the planning process into the two planning phases allows to dramatically reduce the number of web services to be considered and so the number of inquires for offers.

This paper deals with the Concrete Planning Problem (CPP), shown to be NP-hard [7]. Our previous works employ several techniques to solve it: a Genetic Algorithm (GA) [8], numeric optimization methods [9] as well as Satisfiability Modulo Theories (SMT) Solvers [7]. 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 often demonstrates in 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 could be far from optimum and are found with a low probability. Thus, our aim is to exploit the advantages of both methods by combining them into a hybrid algorithm. This methodology and its implementation is the main contribution of the paper. We present here three versions of a hybrid algorithm. Two of them involve a modification of the standard GA in such a way that after every couple of iterations of GA, several top-ranked individuals are processed by the SMT-based algorithm in order to improve them [10]. The third one exploits an SMT-solver in order to generate the initial populations for GA, which results in a substantial improvement of the algorithm efficiency. Such an approach is novel, whereas several other "pure" and hybrid approaches to CPP have been defined. We discuss them in the next section.

II. RELATED WORK

Over the last few years, the concrete planning problem has been extensively studied in the literature. G. Canfora et al. [11] use a simple GA to obtain a good quality concrete plan. As optimization criteria the authors choose features commonly referred to as Quality of Service (QoS), like the response time of a web service, its cost, availability, and reliability. An individual of GA representing a concrete plan is encoded as a vector of integer values, where each value identifies an offer and each position of the vector corresponds to a service type. While a concrete plan has to satisfy a number of constraints, the concrete planning problem is transformed to the constrained optimisation problem. The authors define a penalty function, which decreases the fitness values of the individuals not satisfying some constraints. Unfortunately, experimental study concerns only 25 services and up to 25 offers for each service. However, our approach, where the user is free to define an objective function using any of the available attributes, seems to be much more flexible.

Y. Wu et al. [12] transform CPP to a multi-criteria optimization problem and exploit GA to find a concrete plan. However, the authors present the experiments on a relatively small search space that could not provide valuable conclusions. Another version of GA is presented in [13], where special genetic operators are applied in order to find a concrete plan satisfying user constraints. Moreover, the individuals are generated basing on a set of initially found concrete plans using greedy heuristics. This idea allows to evolve only feasible individuals within the population and the algorithm must only assure that genetic operators do not provide unfeasible potential solutions. Thus, the idea of generating the initial population is somewhat similar to the one applied in our InitPop Hybrid.

Solving CPP using GA with a new version of crossover and mutation operators is presented in [14]. An adaptive crossover and a mutation operator are designed to increase convergence to a local minimum. Moreover, the idea of tabu list, which comes from the Tabu Search algorithm is applied. The experiments show that an applictaion of the modified version of GA gives better results than using the standard one.

Besides many papers discussing an application of GA to solve CPP [15][16][17], there are also papers applying more advanced algorithms. The authors of the paper [18], where CPP is viewed as a multi-criteria optimization problem, use the NSGA-II algorithm to obtain a set of good quality plans. Again, the attributes like the cost, the execution time, the service availability and a reputation stand for the optimization criteria. Instead of defining one fitness function with weights for each feature, the authors propose four separate fitness functions, each for a single attribute. Each plan obtained is optimal according to one of the fitness functions defined in the algorithm. Unfortunately, this result has been obtained thanks to considering small state spaces only, generated by 3 types of services and 15 instances of each type.

Different approaches of the Artificial Immune Systems have also been applied to solve CPP. In [19] a modified version of the CLONALG algorithm is used to a single-phase planning. The algorithm finds simultaneously abstract and concrete plans. Another modification of CLONALG is presented in [20]. At the beginning of the algorithm antibodies representing potential solutions are generated randomly, but it is assured that all of them are feasible. Contrary to the other works, in this algorithm an individual is encoded in a binary way. Similarly to a number of other approaches the authors use QoS features as the optimisation criteria.

Hybrid algorithms for WSC are also known in the literature. In [21] a modified version of the Particle Swarm Optimization algorithm has been used as a method for solving CPP. In this algorithm two additional genetic operators, namely crossover and mutation, are applied to obtain better results than in the standard algorithm.

Another approach consists in a combination of two evolutionary algorithms, Tabu Search and GA [22]. Similarly to ours, the experiments have been performed using randomly generated benchmarks. The authors examine the performance of the hybrid algorithm in comparison with the "pure" GA and Tabu Search. Although, this hybrid method finds good quality concrete plans, our hybrid algorithm allows for dealing with much larger search spaces. A hybrid approach based on an application of GA and the ant algorithm was proposed in [23]. The problem has been transformed to searching for the shortest path in a graph. 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 the efficiency of this approach.

Thus, after a thorough analysis of the literature we can state that the main novelty of our approach consists not only in combining GA with SMT for solving CPP, but also in providing experimental results for benchmarks of very large state spaces.

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

III. PLANICS FRAMEWORK

In this section we give an overview of the PlanICS framework. First, we briefly sketch the main concepts and the consecutive planning phases in order to eventually focus on the concrete planning stage. Then, we give all the definitions necessary to formulate the concrete planning problem. The section ends with an example planning scenario.

A. Overview of PlanICS

An ontology contains a set 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 s *transforms* a world w into another world w'. A *transformation sequence* $s_1 \ldots s_n$ is a sequence of services for which there is a sequence of worlds $w_0 \ldots w_n$ such that s_i transforms w_{i-1} into w_i, for each $1 \leq i \leq n$. The types of services available for planning are defined as elements of the branch of classes rooted at the *Service* concept. Each service type stands for a description of a set of real-world services of similar functionality.

The main goal of the system is to find a composition of services that satisfies a user query. The query interpretation results in two sets of worlds: the initial and the expected ones. Moreover, the query may include additional constraints, especially *quality constraints*, the sum of which is used to choose the best solution from all the potential solutions. Thus, the task of the system is to find such a set of services, which transform some initial world into a world matching some expected one in such a way that the value of the quality function is maximized. Figure 1 shows the general PlanICS architecture. The bold arrows correspond to computation of a plan, the thin arrows model the planner infrastructure, while the dotted arrows represent the user interactions.

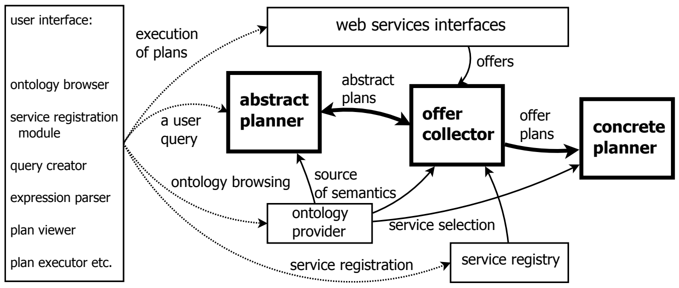

Figure 1. A diagram of the PlanICS system architecture.

The abstract planning is the first stage of the composition in the PlanICS framework. It consists in matching services at the level of input/output types and the abstract values. That is, since 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*. This final world has to be consistent with an expected world, which is 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 [6]. Intuitively, a final world W_f is compatible with an expected world W_e if the following conditions are satisfied:

- 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,
- 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 the 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 [6][24] for more details.

B. Concrete Planning Problem

In the second planning stage, a CAP is used by the Offer Collector, i.e., a tool, which in cooperation with the service registry, queries real-world services. The service registry keeps an evidence of real-world web services, registered accordingly to the service type system. During the registration, the service provider defines a mapping between the input/output data of the real-world service and the object attributes processed by the declared service type. OC communicates with the real-world services of types presented in a CAP. It sends the constraints on the data, which can potentially be sent to the service, and on the data expected to be received in an offer. Usually, each service type represents a set of real-world services. Moreover, querying a single service can result in a number of offers. Thus, we define offer sets as the main result of the second planning stage.

Definition 1 (Offer, Offer set): Assume that the n-th instance of a service type from a CAP processes some number of objects having in total m attributes. A single offer collected by OC is a vector $P = [v_1, v_2, \ldots, v_m]$, where v_j is a value of a single object attribute processed by n-th service of the CAP. An offer set O^n is a $k \times m$ matrix, where each row corresponds to a single offer and k is the number of offers in the set. Thus, the element $o^n_{i,j}$ is the j-th value of the i-th offer collected from the n-th service type instance from the CAP.

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 $\mathbb{O} = (O^1, \ldots, O^n)$ be the vector of offer sets collected by OC such that for every $i = 1, \ldots, n$
$O^i = \begin{bmatrix} o^i_{1,1} & \ldots & o^i_{1,m_i} \\ \vdots & \ddots & \vdots \\ o^i_{k_i,1} & \ldots & o^i_{k_i,m_i} \end{bmatrix}$, and the j-th row of O^i is denoted by P^i_j. Let $\mathbb{P}$ denote the set of all possible sequences $(P^1_{j_1}, \ldots, P^n_{j_n})$, such that $j_i \in \{1, \ldots, k_i\}$ and $i \in \{1, \ldots, n\}$. The Concrete Planning Problem is defined as:

$$max\{Q(S) \mid S \in \mathbb{P}\} \text{ subject to } \mathbb{C}(S), \quad (1)$$

where $Q : \mathbb{P} \mapsto \mathbb{R}$ 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 1. Consider a simple ontology describing a fragment of some financial market consisting of service types inheriting

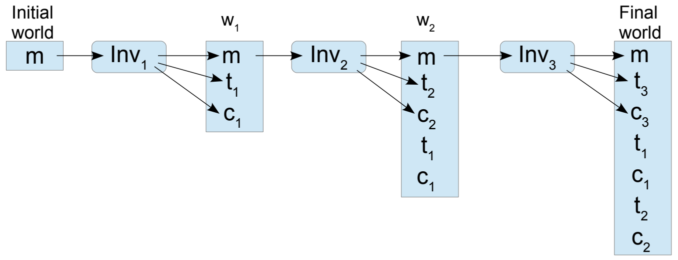

Figure 2. A graphical illustration of the context abstract plan given in the example.

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 $Charge$, respectively, and updates the amount of money remaining after the operation, i.e., the attribute $m.amount$.

Assume that a user would like to invest up to \$100 in three financial instruments, locating 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.

Formally, the user query can be formulated as follows: $in = \emptyset$, $inout = \{m : Money\}$, $out = \{t_1, t_2, t_3 : \text{Transaction}\}$ pre= (m.amount $\leq$ 100), $post$ = (t_1.amount + t_2.amount $>$ 50), $Qual = \{(Transaction, true, profit, Sum)\}$, $Aggr = \{(Charge, true, fee, Max, <, 3)\}$.

The meaning of the consecutive components of the query is the following: in is a set of the read-only objects, while $inout$ is a set of the objects that could be modified - both are available at the start of the composition; out is a set of the objects that do not exist in the initial worlds, but they have to be produced by services of the plan. Next, pre and $post$ are boolean formulae describing the states of the initial and the expected worlds, respectively. Finally, $Qual$ and $Aggr$ are sets of quality constraints, and additional aggregation constraints, respectively.

Consider an exemplary CAP consisting of three instances of the $Investment$ service type depicted in Figure 2. The boxes represent worlds, the round boxes are services, while the arrows stand for transformation contexts.

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: $C_1 := (o^1_{i_1,1} \leq 100)$ and $C_2 := (o^1_{i_1,2} + o^2_{i_2,2} > 50)$, 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: $C_3 := (o^1_{i_1,5} = o^2_{i_2,1})$ and $C_4 := (o^2_{i_2,5} = o^3_{i_3,1})$. The aggregate condition is translated to the following constraint: $C_5 := \big(max(\{o^1_{i_1,4}, o^2_{i_2,4}, o^3_{i_3,4}\}) < 3\big)$, while the quality expression is translated to the quality constraint $Q := \sum_{j=1}^{3} o^j_{i_j,3}$.

IV. Hybrid Solutions

So far, we have made the following observations of the experiments with the "pure" SMT- and GA-based planners. The main disadvantage of the 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, the GA-based approach is relatively fast, but it yields solutions that are far from optimum, and of low probability. Thus, our strategy is to delegate some sub-problems to be solved by an SMT-solver in such a way that the computation time is acceptable while the results allow to obtain better performance of GA. In order to evaluate the efficiency of our new hybrid algorithms, we use as benchmarks several CPP instances, which can hardly be solved by "pure" planners. By comparing the performance of several versions of the hybrid algorithm (with various parameter combinations) on the same examples, we can conclude whether the hybrid algorithm outperforms each of the "pure" methods separately.

In this section we present the main ideas behind three hybrid algorithms, named: Random Hybrid (RH) [1], Semi-Random Hybrid (SRH) [10], and InitPop Hybrid (IPH). We start with a description of their common features.

A. Overview

Our hybrid approach is based on 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 Definition 2), multiplied by some constant β:

$$fitness(I) = q(S_I) + \beta \cdot \frac{|sat(\mathbb{C}(S_I))|}{c}, \tag{2}$$

where I stands for an individual, S_I is a sequence of the offer values corresponding to I, $sat\big(\mathbb{C}(S_I)\big)$ 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 β 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

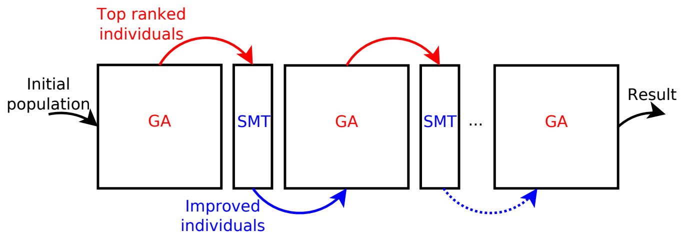

Figure 3. The RH and SRH algorithm overview.

final result. The value of β depends on the estimation of the minimal and the maximal quality function value.

B. Random and Semi-Random Hybrid Algorithms

The RH and SRH algorithms are based on the following modification of the standard GA (see Figure 3). 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 that randomly selects the genes to be changed.

The SMT procedure combined with GA is based on the encoding exploited in our "pure" SMT-based concrete planner [7][9]. The idea is to encode *SFII* as an SMT formula that is satisfiable if such an individual exists. First, we initialize an SMT-solver allocating a set $\mathcal{V}$ of all necessary variables:

- $\mathbf{oid}^i$, 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 $\mathbf{oid}^i$ variable takes a value between 1 and k_i.
- $\mathbf{o}^i_j$, 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 that 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 $\mathcal{V}$, 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 $\mathbb{O} = (O^1, \ldots, O^n)$ are encoded as the formula

$$ofr(\mathbb{O}, \mathcal{V}) = \bigwedge_{i=1}^{n} \bigvee_{d=1}^{k_i} \Big(\mathbf{oid}^i = d \;\wedge \bigwedge_{o^i_{d,j} \in R^i} \mathbf{o}^i_j = o^i_{d,j}\Big). \quad (3)$$

The formulae $ctr\big(\mathbb{C}(S), \mathcal{V}\big)$ and $qual\big(Q(S), \mathcal{V}\big)$, denoted as $\mathbf{ctr}$ and $\mathbf{q}$ for short, encode the constraints and the objective function, respectively. Details are provided in [7].

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:

$$\bigwedge_{i \in \{1,\ldots,n\} \setminus M} (\mathbf{oid}^i = g_i) \wedge ofr\big(\mathbb{O}, \mathcal{V}\big) \wedge \mathbf{ctr} \wedge (\mathbf{q} > q(S_I)) \quad (4)$$

That is, the Formula (4) is satisfiable only if there exists an individual $I' = (g'_1, \ldots, g'_n)$ satisfying all the constraints, where $\forall_{i \notin M}\ g_i = g'_i$ 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.

Although, we have presented the general idea of a hybrid algorithm, there are still a number of problems that need to be solved in order to combine GA and SMT. Moreover, they can be solved in many different ways. The crucial questions that need to be answered are as follows. When to start the *SFII* procedure for the first time? How many times and how often *SFII* should be run? How many genes should remain fixed? How to choose genes to be changed? Since there are many possibilities to deal with the above problems, we started from the simplest solution that randomly selects genes to be changed. The solutions to the remaining questions we treat as parameters in order to develop the first version of our hybrid solution, called Random Hybrid (RH). Its pseudo-code is presented in Algorithm 1.

After analysing the experimental results (see Section V) we have found that the results are slightly better than these obtained using GA and SMT separately, however they could still be improved, especially in terms of a higher probability and a lower computation time. Thus, we introduced several improvements to the RH algorithm and we implemented the Semi-Random Hybrid (SRH) algorithm. The most important improvements introduced to SRH are as follows.

The $selectGenes$ procedure (see the line 11 of Algorithm 1) is not completely random any more. In the first place the genes violating some constraints are chosen to be changed. Then, the additional gene indices are selected randomly until we get a set of size gn. This change allows to increase the probability of finding a solution.

The next improvement aims at reducing the computation time. It consists in running the *SFII* procedure only if an individual violates some constraints. Thus, in case of SRH the lines from 11 to 15 in Algorithm 1 are executed conditionally, only if the individual I violates some constraints. In Section V we discuss the results obtained and compare both the approaches with the third hybrid solution - the IPH algorithm.

C. The InitPop Hybrid Algorithm (IPH)

The third hybrid algorithm also combines GA with SMT, but it does it in a different way. Our observations of the behaviour of the RH and SRH algorithms have had a big influence on the third hybrid algorithm. The first conclusion is

```
1  Procedure RandomHybrid($st$, $ind$, $int$, $gn$, $N$)
   Input: $st$: when to start SFII for the first time,
   $ind$: the number of individuals to pass to SFII during a single GA iteration,
   $int$: how often run SFII,
   $gn$: the number of genes to be changed by SFII,
   $N$: the number of GA iterations.
   Result: an individual representing the best concrete plan
           found, or $null$
2  begin
3      $initialize()$; // generate initial population, initialize SMT solver
4      $evaluate()$; // compute fitness function for all individuals
5      for ($i \leftarrow 1; i \leq N; i \leftarrow i+1$) do
6          $selection(); crossover(); mutation()$; // ordinary GA routines
7          $evaluate()$;
8          if ($i \geq st$) $\wedge$ ($i$ mod $int = 0$) then
9              $BI \leftarrow findBestInd(ind)$; // a set of $ind$ top individuals
10             foreach $I \in BI$ do
11                 $M \leftarrow selectGenes(I, gn)$; // a set of gene indices to be changed
12                 $I' \leftarrow runSFII(M)$;
13                 if $I' \neq null$ then
14                     $I \leftarrow I'$; // replace $I$ by $I'$ in the current population
15                 end
16             end
17         end
18     end
19     $\{best\} \leftarrow findBestInd(1)$;
20     if $constraintsSatisfied(best)$ then
21         return $best$; // if a valid solution has been found
22     else
23         return $null$
24     end
25 end
```

Algorithm 1: A pseudocode of the RandomHybrid algorithm

that the larger number of constraints, the worse performance of GA. However, a large number of constraints does not worsen the efficiency of the SMT-based components. On the other hand, searching for individuals of quality higher than a given value is quite expensive for an SMT-solver, while this is a natural application of GA. Therefore, the main idea is to divide the tasks of both the modules so that each of them is doing its best.

Now, the SMT-solver is used to generate (a part of) the initial population for GA (see Figure 4). The individuals generated by the SMT-solver satisfy all constraints. Note that each such individual is already a solution of CPP, but typically its

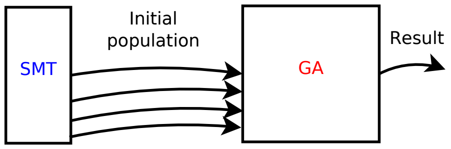

Figure 4. The IPH algorithm overview.

fitness value is not optimal. However, the individuals generated by an SMT-solver provide an "easy start" for the genetic algorithm. Moreover, since the initial population already contains at least one solution, the algorithm should *always* return a plan. On the other hand, if the collected offers do not allow to build a solution (or the constraints are contradictory), then we get a straight negative answer from the SMT-solver.

We have chosen a simple, but fast, strategy of generating individuals. That is, an individual is a valuation of the $\mathbf{oid}^1$, $\ldots$, $\mathbf{oid}^n$ variables satisfying the conjunction of the formula encoding the offers and the formula encoding the constraints: $ofr(\mathbb{O}, \mathcal{V}) \wedge \mathbf{ctr}$. Thus, in this case the sub-problem solved by the SMT-solver is much simpler than in the RH and SRH algorithms. In order to generate more than one individual, we also construct a formula blocking all valuations previously found, and this formula is conjuncted with the formula passed to the SMT-solver. That is, in order to generate the i-th individual the SMT-solver has to check the satisfiability of the following formula:

$$\varphi_i = ofr(\mathbb{O}, \mathcal{V}) \wedge \mathbf{ctr} \wedge \mathbf{B}_{i-1}. \qquad (5)$$

Here, $\mathbf{B}_i$ stands for a blocking formula, which is defined inductively as:

$$\begin{aligned} \mathbf{B}_0 &= true, \\ \mathbf{B}_{i+1} &= \mathbf{B}_{i-1} \wedge \neg\Big(\bigwedge_{j=1..n} \mathbf{oid}^j = val_i(\mathbf{oid}^j)\Big), \end{aligned} \qquad (6)$$

where $val_i(\mathbf{oid}^j)$ is the value of the j-th gene returned by the solver in the i-th step.

In the next section we provide the results of several experiments and compare the results of all three hybrid algorithms.

V. Experimental Results

In our previous experiments, discussed in [9], we have applied several "pure" methods to the concrete planning. These methods include the SMT-based approach, the standard GA, and numeric algorithms implemented in the OpenOpt toolset. The results of our experiments can be summarised as follows. The SMT-based planner is always able to find the optimal solution, provided that it is enough time and memory. In contrast, GA can find sometimes the optimal solution of length at most 5, but it consumes less time and memory. The ability of GA to find a concrete plan depends on the number of constraints. The more optimization constraints the smaller probability of finding a concrete plan. These drawbacks of GA are not common to our SMT-based approach. Moreover, our experiments show that a large number of constraints helps the SMT-solver to reduce the search space and to find the optimal solution faster. Overall, both the approaches are complementary and behave differently depending upon a particular problem instance. Concerning OpenOpt, we have shown that it can also be used for solving CPP. However, its effectiveness is satisfactory only if no tuples of values are present in the problem domain and much worse in the opposite case.

In order to evaluate the efficiency of our hybrid algorithms on "difficult" benchmarks, we have used for the experiments six instances of CPP that have been hardly solved with our "pure" SMT- and GA-based planners [7]. All the instances represent plans of length 15. Each offer set of Instance I,

TABLE I. Experimental results for Instances I and II.

				INSTANCE I						INSTANCE II					
				Random			Semi-Random			Random			Semi-Random		
exp	gn	ind	int	t[s]	Q	P	t[s]	Q	P	t[s]	Q	P	t[s]	Q	P
1	8	1	10	9.3	1305.0	3.3	9.3	1271.2	16.7	14.9	1382.0	6.7	15.9	1323.1	26.7
2			20	8.2	1331.5	6.7	7.9	1303.2	30.0	13.2	1371.5	13.3	12.3	1386.3	20.0
3		10	10	41.0	1386.7	53.3	25.7	1349.3	73.3	59.5	1437.6	36.7	40.0	1367.9	63.3
4			20	22.4	1389.0	26.7	17.9	1313.4	46.7	41.7	1414.0	33.3	31.4	1375.4	60.0
5		20	10	76.3	1405.8	70.0	36.4	1351.2	86.7	118.1	1441.0	73.3	66.3	1390.6	83.3
6			20	34.3	1356.5	43.3	24.9	1325.7	73.3	61.9	1420.3	40.0	43.5	1396.3	70.0
7	12	1	10	39.6	1363.1	66.7	**19.5**	**1337.7**	**86.7**	**56.6**	**1405.3**	**93.3**	30.0	1387.5	80.0
8			20	**14.5**	**1326.9**	**46.7**	11.0	1332.5	43.3	20.4	1380.0	40.0	**17.4**	**1369.9**	**73.3**
9		10	10	203.6	1417.6	100	74.8	1373.1	100	273.2	1455.8	100	108.1	1411.3	100
10			20	114.7	1362.2	100	54.0	1356.8	100	155.9	1431.3	100	76.5	1405.9	100
11		20	10	346.5	1424.2	100	122	1383.9	100	443.1	1460.5	100	166.4	1431.2	100
12			20	196.4	1416.5	100	71.9	1374.1	100	261.7	1455.3	100	96.9	1408.4	100
SMT				266.0	1443.0	100				388.0	1467.0	100			
GA				5.0	1218.5	8.0				5.6	1319.9	10.0			

TABLE II. Experimental results for Instances III and IV.

				INSTANCE III						INSTANCE IV					
				Random			Semi-Random			Random			Semi-Random		
exp	gn	ind	int	t[s]	Q	P	t[s]	Q	P	t[s]	Q	P	t[s]	Q	P
1	8	1	10	13.0	2176.5	6.7	10.5	2077.4	16.7	22.1	2229.5	6.7	19.7	2124.4	33.3
2			20	12.4	2054.3	10.0	10.9	2144.6	16.7	22.0	2193.6	16.7	16.0	2141.2	16.7
3		10	10	121.7	2311.5	46.7	54.8	2217.8	83.3	248.3	2359.1	43.3	101.0	2226.6	70.0
4			20	54.2	2279.4	26.7	**27.6**	**2217.8**	**83.3**	**151.9**	**2353.5**	**43.3**	58.7	2224.4	53.3
5		20	10	324.9	2369.4	76.7	94.3	2284.3	76.7	566.8	2390.7	60.0	195.4	2284.8	90.0
6			20	175.7	2304.2	50.0	58.3	2233.0	70.0	290.8	2334.1	53.3	89.2	2215.2	73.3
7	12	1	10	208.1	2153.4	46.7	55.8	2205.6	90.0	239.7	2216.3	56.7	92.9	2223.2	83.3
8			20	**54.1**	**2274.1**	**36.7**	43.8	2131.7	53.3	64.1	2167.0	26.7	**55.8**	**2226.3**	**60.0**
9		10	10	1727.1	2377.9	100	327.3	2282.2	100	2205.2	2485.3	100	553.7	2328.8	100
10			20	1066.5	2327.7	96.7	213.0	2286.5	100	1325.1	2414.3	96.7	291.6	2246.1	96.7
11		20	10	2814.4	2447.1	100	650.8	2364.7	100	4455.6	2568.2	100	882.4	2408.3	100
12			20	2027.1	2387.3	100	337.8	2317.8	100	2477.0	2469.8	96.7	416.9	2338.2	100
SMT				500.0	2266.0	100				500.0	2409.0	100			
GA				6.0	2085.4	10.0				6.6	2001.9	7.0			

TABLE III. Experimental results for Instances V and VI

				INSTANCE V						INSTANCE VI					
				Random			Semi-Random			Random			Semi-Random		
exp	gn	ind	int	t[s]	Q	P	t[s]	Q	P	t[s]	Q	P	t[s]	Q	P
1	8	1	10	12.0	560.0	36.7	16.3	546.1	66.7	25.4	584.1	63.3	20.5	572.1	100
2			20	8.8	486.0	16.7	10.8	509.5	66.7	17.4	585.3	33.3	16.4	564.3	90.0
3		10	10	55.0	704.1	93.3	34.1	648.0	96.7	156.1	804.8	100	81.9	699.9	100
4			20	**36.2**	**638.2**	**80.0**	22.7	596.1	93.3	**96.8**	**722.8**	**93.3**	**40.6**	**660.1**	**96.7**
5		20	10	94.9	777.9	96.7	51.8	687.1	96.7	298.7	888.1	100	131.8	783.0	100
6			20	52.0	667.3	83.3	**28.2**	**634.1**	**100**	165.9	808.9	100	81.5	694.4	100
7	12	1	10	69.8	620.2	96.7	31.8	569.6	100	124.4	660.3	100	42.2	546.8	100
8			20	30.4	561.1	93.3	23.0	532.6	96.7	75.1	608.8	90.0	42.5	588.6	96.7
9		10	10	420.0	852.8	100	139.9	731.4	100	1385.4	928.5	100	294.7	769.0	100
10			20	264.1	774.5	100	66.5	620.8	100	619.6	814.4	100	122.3	628.9	100
11		20	10	807.4	935.5	100	275.7	832.7	100	2614.5	993.3	100	643.2	874.4	100
12			20	461.9	852.6	100	100.7	675.7	100	1464.6	927.3	100	260.1	750.9	100
SMT				500.0	781.0	100				500.0	755.0	100			
GA				5.1	436.0	8.0				5.9	537.8	12.0			

III, and V contains 256 offers, which makes the number of the potential solutions equal to $256^{15} = 2^{120}$. In the case of Instance II, IV, and VI, each offer set consists of 512 offers, which results in the search space size as large as $512^{15} = 2^{135}$. The objective functions used for the Instances from I to IV are as follows:

$$Q_{1,2} = \sum_{i=1}^{n} o^{i}_{j_i,1}, \tag{7}$$

$$Q_{3,4} = \sum_{i=1}^{n} (o^{i}_{j_i,1} + o^{i}_{j_i,2}). \tag{8}$$

The set of the constraints of the Instances from I to IV is defined as follows:

$$\mathbb{C}_{1,2,3,4} = \{(o^{i}_{j_i,2} < o^{i+1}_{j_{i+1},2}) \mid i = 1, \ldots, n-1\}. \tag{9}$$

That is, for the Instances I and II our aim is to maximize a sum of n values, while for the Instances III and IV the sum of $2n$ values is to be maximized. Since the concrete planning is reduced to the constrained optimisation problem and it is clearly separated from the previous planning stages, the concrete planners do not need to “know” what the planning domain is, and what the particular variables mean. This allowed us to generate the instances randomly using our software Offer Generator.

In the case of the Instance V and VI, which are based on Example 1, the objective functions and the constraints are as follows:

$$Q_{5,6} = \sum_{i=1}^{n} o^{i}_{j_i,3}, \quad (10)$$

$$\mathbb{C}_{5,6} = \{(o^{1}_{j_1,1} \leq 100), (o^{1}_{j_1,2} + o^{2}_{j_1,2} > 50), (o^{i}_{j_i,5} = o^{i+1}_{j_{i+1},1}) \mid i = 1, \ldots, n-1\}. \quad (11)$$

This time we can provide a clear interpretation of the objective function and the constraints, as they follow the user query given in the example. The objective functions Q_5 and Q_6 correspond directly to the function Q defined in Example 1. Since the third value of an offer corresponds to the $profit$ attribute, the sum of the profits is to be maximized here. Similarly, the constraint sets also correspond to these defined in Example 1.

Besides the parameters introduced already in Section IV, the standard parameters of GA, used in the hybrid algorithms, have been set to the same values as in the pure GA, that is, the population size – 1000, the number of iterations – $N = 100$, the crossover probability – 95%, and the mutation probability – 0.5%. Moreover, all the experiments with the hybrid algorithms have been performed using $st = 20$, that is, the first *SFII* procedure starts with the 20th iteration. Every instance has been tested 12 times, using a different combination of the remaining parameter values (see Tables from I to III), and every experiment has been repeated 30 times on a standard PC with 2.8GHz CPU and Z3 [25] version 4.3 as the SMT-solving engine.

The results of applying the RH and SRH algorithms to Instances I - VI are presented in Tables I, II, and III, where the columns from left to right display the experiment label, the parameter values, and for each Instance and each hybrid variant the total runtime of the algorithm (**t[s]**), the average quality of the solutions found (**Q**), and the probability of finding a solution (**P**). For reference, we report in the two bottom rows (marked with SMT and GA, respectively) the results of the pure SMT- and GA-based planner. The pure GA-based planner was run with the same parameters values as the hybrid ones. The test has been performed on the same machine.

One can easily see that the quality values obtained in almost every experiment are higher than these returned by GA. However, in several cases either the runtime or the probability is hardly acceptable. On the other hand, for many parameter combinations we obtain significantly better results in terms of the runtime (comparing to the pure SMT) or the probability (in comparison with the pure GA). We marked in bold the results that we find the best for a given instance and a hybrid variant.

Although the results are very promising and encouraging, as one could expect, the hybrid solutions are clearly a trade-off between the three measures: the quality, the probability, and the computation time of the pure algorithms. It is easy to observe that for many parameter valuations the hybrid algorithms outperform each pure planning method provided one or two measures are taken into account only. Moreover, the Semi-Random Hybrid algorithm outranks in almost all cases the Random Hybrid one in terms of the computation time and the probability of finding a solution. On the other hand, since RH runs SMT-solver much more often than SRH, it also finds solutions of better quality than SRH, but at the price of a much longer computation time.

In order to compare the results obtained taking all the three measures into account at the same time, we defined two simple score functions:

$$score_i(P, t, Q) = \frac{P}{t} \cdot (Q - const_i)$$

$$score_{p^2}(P, t, Q) = \frac{P^2 \cdot Q}{t}, \quad (12)$$

where P, t, and Q stand for the probability, the computation time, and the average quality, respectively, and $const_i$ is a parameter, which value is selected for each Instance i from I to VI, to make the scores of the pure GA- and SMT-based algorithm equal. The values of the $const_i$ parameters used for comparing the results for Instances I-VI are as follows: 1150, 1295, 2061, 1906, 386, 514, respectively.

These scores are then selected as the benchmarks for the comparison given in Figure 5. The dark- and light-grey bars correspond to the results obtained with the RH and SRH algorithm, respectively.

The $score_i$ function aims at comparing the results under the assumption that both the pure planning methods are equally effective as far as the three measures are concerned. On the other hand, the $score_{p^2}$ function gives priority to the solutions having a high probability. Obviously, this way, one can define a number of other score functions in order to compare the results according to a personal preference. Notice that another interesting remark can be made about the hybrid parameter values. Namely, the bold values in Tables I, II, and III, as well as the highest chart bars in Figure 5 most often correspond to parameter combinations of the experiment 4, 7, and 8. However, the study of only six instances does not allow us to draw any broad conclusions. Therefore, in our future work we are going to investigate whether these parameter values guarantee to obtain good results in general.

Next, we use the same benchmarks in order to evaluate the efficiency of the IPH algorithm. The results are presented in Table IV. Its first column contains the number of the individuals in the initial population generated by an SMT-solver. While the total population size is equal to 1000, the remaining individuals are created randomly.

The data in the next columns stand for the computation time and the quality of solutions found given for each instance from I to VI.

The most important advantage of the IPH algorithm over the other algorithms is that the probability of finding solutions in all cases is equal to 100%. This is the consequence of generating at least one individual, which is already a solution at the start of GA. Also, the computation time of IPH is much smaller than in the case of the pure SMT as well as the RH and SRH algorithms. On the negative side of the IPH algorithm, the quality function value of the solutions found is lower.

The comparison of the results of all three algorithms based on values of the score functions in shown in Figure 6. At the X axis: $1, 100, 500, 1000$ are the numbers of the individuals generated by the SMT-solver in the initial population; SMT and GA stand for the results obtained using the respective "pure" planning methods while RH and SRH denote the best results obtained with Random and Semi-Random Hybrid algorithms.

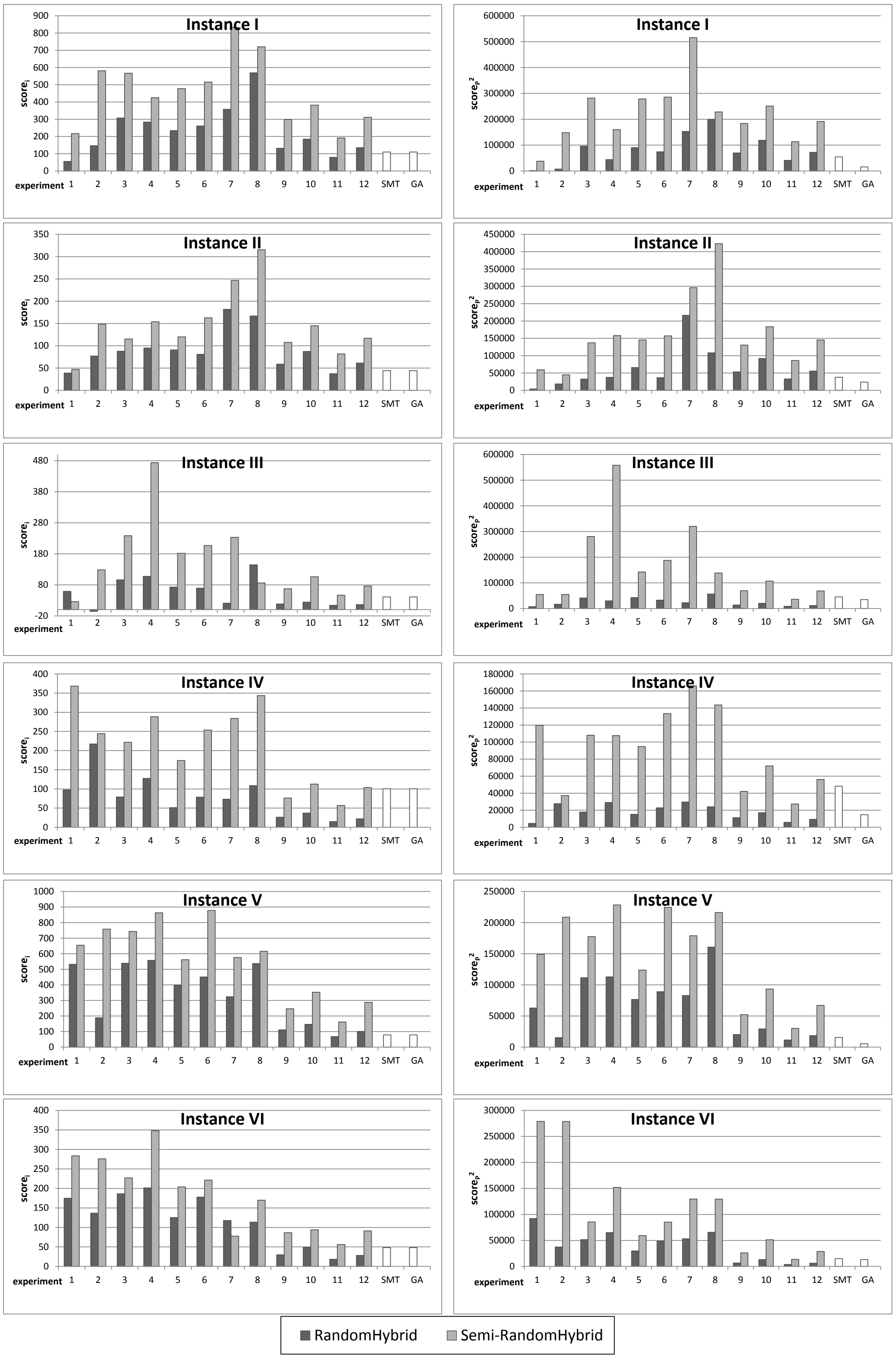

Figure 5. The comparison of the RH and SRH algorithm performance based on two score functions.

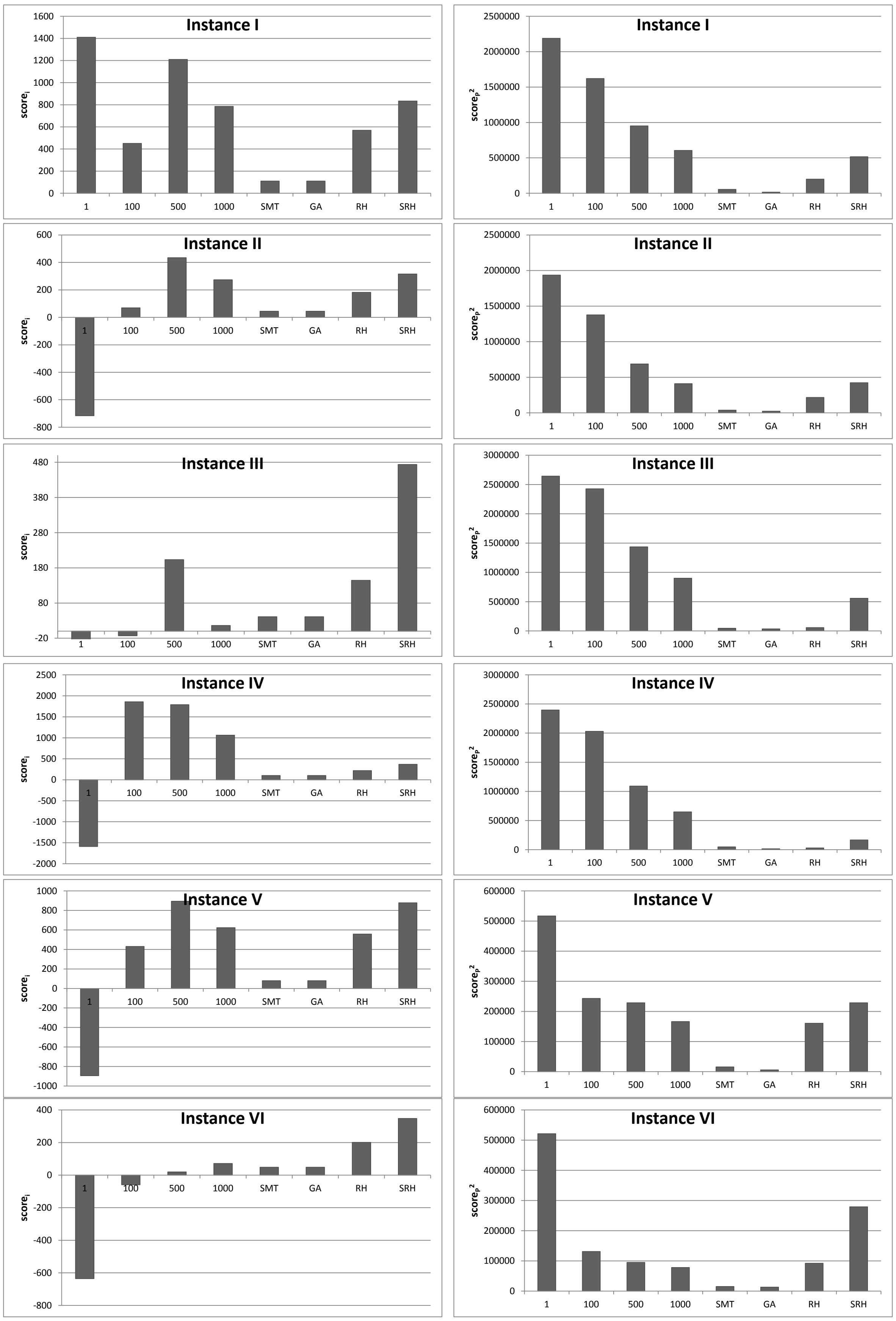

Figure 6. The comparison of the results of the three algorithms based on the values of the score functions.

TABLE IV. EXPERIMENTAL RESULTS OF THE IPH ALGORITHM.

	INSTANCE I		INSTANCE II		INSTANCE III	
inds	t[s]	Q	t[s]	Q	t[s]	Q
1	5,6	1229,5	6,4	1248,8	6,4	1706,8
100	7,3	1183,2	9,5	1301,5	8,4	2059,9
500	13,9	1317,9	20,1	1382,4	14,6	2090,6
1000	21,8	1321,7	33,8	1387,4	23,0	2064,7

	INSTANCE IV		INSTANCE V		INSTANCE VI	
inds	t[s]	Q	t[s]	Q	t[s]	Q
1	7,5	1788,0	6,4	329,1	8,8	458,2
100	10,3	2098,5	19,3	469,2	37,6	491,7
500	20,9	2280,3	27,8	634,5	55,2	524,7
1000	35,2	2280,3	37,2	618,0	72,5	565,6

VI. CONCLUSION AND FUTURE WORK

In this paper three versions of the hybrid concrete planner have been implemented and several experiments have been performed. The experimental results show that even when using a straightforward strategy of combining the SMT- and GA-based approach, one can obtain surprisingly good results. We believe that all of the proposed methods are of high potential. However, the most promising approach seems to be the InitPop Hybrid algorithm, due to its relatively low computation time and the ability to always return a solution, if there exists one. These advantages follow from dividing the tasks of both the combined methods so that each of them is doing its best. While the SMT-solver deals easily even with a large number of constraints, the genetic algorithm copes with the objective functions.

An important task to be addressed in a future work will consist in investigating how to choose the parameter values in order to get a trade-off between the quality, the 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

[1] 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.

[2] M. Bell, Introduction to Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture. John Wiley & Sons, 2008.

[3] S. Ambroszkiewicz, “Entish: A language for describing data processing in open distributed systems,” Fundam. Inform., vol. 60, no. 1-4, 2003, pp. 41–66.

[4] 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.

[5] D. Doliwa et al., “PlanICS - a web service compositon toolset,” Fundam. Inform., vol. 112(1), 2011, pp. 47–71.

[6] 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.

[7] A. Niewiadomski, W. Penczek, and J. Skaruz, “SMT vs genetic algorithms: Concrete planning in PlanICS framework,” in CS&P, 2013, pp. 309–321.

[8] J. Skaruz, A. Niewiadomski, and W. Penczek, “Automated abstract planning with use of genetic algorithms,” in GECCO (Companion), 2013, pp. 129–130.

[9] A. Niewiadomski, W. Penczek, J. Skaruz, M. Szreter, and M. Jarocki, “SMT versus Genetic and OpenOpt Algorithms: Concrete Planning in the PlanICS Framework,” (accepted to Fundam. Inform.), 2014.

[10] A. Niewiadomski, W. Penczek, and J. Skaruz, “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] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, “An approach for QoS-aware service composition based on genetic algorithms,” in Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, 2005, pp. 1069–1075.

[12] Y. Wu and X. Wang, “Applying multi-objective genetic algorithms to QoS-aware web service global selection,” Advances in Information Sciences and Service Sciences, vol. 3(11), 2011, pp. 134–144.

[13] M. AllamehAmiri and V. D. dn M. Ghasemzadeh, “Qos -based web service composition based on genetic algotihm,” Journal of AI and Data Mining, vol. 1, 2013, pp. 63–73.

[14] Y. Y. Zhang, H. L. Xiong, and Y. C. Zhang, “An improved genetic algorithm of web services composition with qos,” Advanced Materials Research, vol. 532, 2012, pp. 1836–1840.

[15] T. Weise, S. Bleul, and K. Geihs, “Web service composition systems for the web service challenge – a detailed review,” 2007.

[16] H. Jiang, X. Yang, K. Yin, S. Zhang, and J. A. Cristoforo, “Multi-path QoS-aware web service composition using variable length chromosome genetic algorithm,” Information Technology Journal, vol. 10, 2011, pp. 113–119.

[17] L. Zhang, B. Li, T. Chao, and H. Chang, “On demand web services-based business process composition,” in Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics. IEEE Computer Society, 2003, pp. 4057–4064.

[18] D. B. Claro, P. Albers, and J. kao Hao, “Selecting web services for optimal composition,” in Proc. of the 2nd Int. Workshop On Semantic And Dynamic Web Processes, 2005, pp. 32–45.

[19] I. Salomie, M. Vlad, V. Chifu, and C. Pop, “Hybrid immune-inspired method for selecting the optimal or a near-optimal service composition,” in Proc. of the Computer Science and Information Systems. IEEE Computer Society, 2011, pp. 997–1003.

[20] J. Xu and S. Reiff-Marganiec, “Towards heuristic web services composition using immune algorithm,” in Proc. of the IEEE Conf. on Web Services. IEEE Computer Society, 2008, pp. 238–245.

[21] C. Hu, X. Chen, and X. Liang, “Dynamic services selection algorithm in web services composition supporting cross-enterprises collaboration,” Journal of Central South University of Technology, vol. 16, 2009, pp. 269–274.

[22] 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.

[23] 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.

[24] 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.

[25] 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.