A Hybrid Approach to Web Service Composition Problem in the PlanICS Framework*

Artur Niewiadomski1, Wojciech Penczek1,2, and Jarosław Skaruz1

1 ICS, Siedlce University, 3-Maja 54, 08-110 Siedlce, Poland
{artur.niewiadomski,jaroslav.skaruz}@uph.edu.pl
2 ICS, Polish Academy of Sciences, Jana Kazimierza 5, 01-248 Warsaw, Poland
penczek@ipipan.waw.pl

Abstract. The paper deals with the concrete planning problem – a stage of the Web Service Composition in the PlanICS framework. A novel (hybrid) planning technique based on a combination of a Genetic Algorithm and a Satisfiability Modulo Theories Solver is introduced. The experimental results of the hybrid algorithm are compared with those obtained using “pure” planning methods.

1 Introduction

Service-Oriented Architecture (SOA) [2] exploits the idea of composing simple functionalities, accessible via well-defined interfaces, in order to satisfy more sophisticated objectives. The problem of finding such a composition is hard and known as the Web Service Composition (WSC) problem [1,2,9].

The system PlanICS [4] is a framework aimed at WSC, which allows for 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, the 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, shown to be NP-hard [7]. Our previous papers employ several techniques to solve it: a genetic algorithm (GA) [10], numeric optimization methods [8], and Satisfiability Modulo Theories (SMT) Solvers[7]. The results of our extensive experiments show that the proposed methods are complementary, but every single one suffers from

* This work has been supported by the National Science Centre under the grant No. 2011/01/B/ST6/01477.
some disadvantages. This observation is the motivation to combining the power of SMT with the potential of GA, which is the main contribution of this paper.

The principal 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 of low probability. Thus, our aim is to exploit the advantages of both methods by combining them into one hybrid algorithm. In the paper we present two new hybrid algorithms and compare their efficiency with the pure SMT- and GA-based planner on several benchmarks.

The rest of the paper is structured as follows. In Section 2 the Planics framework is introduced and the Concrete Planning Problem (CPP) is defined. Section 3 presents the main ideas of our hybrid approach as well as some technical solutions. Next, the preliminary experimental results are presented and discussed, followed by conclusions.

2 Concrete Planning Problem

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

![Diagram](image)

Fig. 1. A diagram of the Planics system 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.

An ontology contains a system of classes describing the types of the services as well as the types of the objects they process [6]. 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
A Hybrid Approach to Web Service Composition Problem

service transforms a world. The types of the 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 features.

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 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. Fig.1 shows the general Plantsc architecture.

In the first stage of the composition an abstract planner matches services at the level of input/output types and the abstract values\(^1\) [5]. The result of this stage is a Context Abstract Plan (CAP), consisting of a multiset of service types (defined by a representative sequence), contexts (mappings between services and the objects being processed), and a set of final worlds\(^2\) containing objects that fulfill the user query.

In the second planning stage CAP is used by the offer collector (OC), 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 present in a CAP, sending the constraints on the data, which can potentially be sent to the service in an inquiry, and on the data expected to be received in an offer in order to keep on building a potential plan. 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, for \(1 \leq j \leq m\), \(v_j\) is a value of a single object attribute processed by the \(n\)-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 \(O^n_{ij}\) from \(O^n\) 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

\(^1\) At this planning stage it is enough to know if an attribute does have a value, or it does not, so we abstract from the concrete values of the object attributes.

\(^2\) The user query \(q\) defines a set of initial and expected worlds. A CAP for \(q\) transforms some initial world into a set of final worlds, where at least one of them has to match one of the expected worlds. See [5] for details.
of OG are as follows: (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 \( O = (O^1, \ldots, O^n) \) be the vector of offer sets collected by \( OC \) such that for every \( i = 1, \ldots, n \)

\[
O^i = \left[ \begin{array}{c}
\sigma^i_{1,1} & \cdots & \sigma^i_{1,m_i} \\
\vdots & \ddots & \vdots \\
\sigma^i_{k_i,1} & \cdots & \sigma^i_{k_i,m_i}
\end{array} \right],
\]

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

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

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

### 3 Hybrid Solution

The analysis of several CPP instances, which are hard to solve by "pure" SMT- and GA-based planners, is our main motivation for combining both methods. 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, which are far from optimum and of low probability. Thus, our aim is to exploit the advantages of both the methods by combining them into one hybrid algorithm.

#### 3.1 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 an individual, which, in case of CPP, is a sequence of natural values. In each iteration of GA a set of individuals is selected in order to apply genetic operations such as the standard one-point crossover and mutation. This 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 $\beta$:

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

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 the constraints. The parameter $\beta$ 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$^3$.

The main idea of our new hybrid approach consists in the 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).

### 3.2 Encoding

The SMT-based procedure combined with GA is based on the encoding exploited in our "pure" SMT-based concrete planner $^5$. 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 the set $V$ of all necessary variables:

- $old^i$, for $i = 1...n$, where $n$ is the length of the abstract plan. These variables are used for storing the identifiers of the offers constituting a solution.
- A single $old^i$ variable takes a value from 1 to $k_i$.
- $o_j^i$, for $i = 1...n$, $j = 1...m_i$, where $m_i$ is the number of the offer values in the $i$-th offer set. They are used for encoding the values of $S_i$, 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 $O = (O^1,...,O^n)$ are encoded as the following formula:

$$ofr(O, V) = \bigwedge_{i=1}^{n} \bigvee_{d=1}^{k_i} \big( old^i = d \land \bigwedge_{o_{i,j}^i \in R^i} o_{i,j}^i \big). \quad (3)$$

$^3$ The value of $\beta$ depends on the estimation of the minimal and the maximal quality function value.
The formulae $\text{ctr}(\mathcal{C}(S), \mathcal{V})$ and $\text{qual}(\mathcal{Q}(S), \mathcal{V})$, denoted as $\text{ctr}$ and $\text{q}$ for short, encode the constraints and the objective function, respectively. Due to the space limit the details are omitted here.

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

$$\bigwedge_{i \in \{1, \ldots, n\} \setminus M} (\text{old}^i = g_i) \land \text{ofr}(\emptyset, \mathcal{V}) \land \text{ctr} \land (q > q(S_I))$$ (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 $\forall_{i \not\in M} g_i = g_i'$ and $q(S_{I'}) > q(S_I)$, i.e., sharing with $I$ all genes of the indices outside $M$ and having a higher value of the objective function than $I$. If the formula is satisfiable, then the values of the genes changed are decoded from the model returned by the SMT-solver, and the improved individual $I'$ replaces $I$ in the current population.

3.3 Hybrid Variants

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 often should $SFII$ 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 which 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 analyzing the experimental results (see Section 4) we found that the results are slightly better than those 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 in the RH algorithm and we implemented the Semi-Random Hybrid (SRH) algorithm. The most important improvements introduced in SRH are as follows.

The $\text{selectGenes}$ procedure (see 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, 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 the next section we compare both the approaches and discuss the results obtained.
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 change by SFII, N: the number of GA
iterations
Result: an individual representing the best concrete plan found, or null
begin
3 initialize(); // generate initial population, initialize SMT solver
4 evaluate(); // compute fitness function for all individuals
5 for (i ← 1; i ≤ N; i ← i + 1) do
6 selection(); crossover(); mutation(); // ordinary GA routines
7 evaluate();
8 if (i ≥ st) ∧ (i mod int = 0) then
9 BI ← findBestInd(ind); // a set of ind top individuals
10 foreach I ∈ BI do
11 M ← selectGenes(I, gn); // a set of gene indices to be
12 changed
13 I' ← runSFII(M);
14 if I' ≠ null then
15 I ← I'; // replace I by I' in the current population
16 end
17 end
18 {best} ← findBestInd(1);
19 if constraintsSatisfied(best) then
20 return best; // if a valid solution has been found
21 else
22 return null
23 end
24 end

Algorithm 1. Pseudocode of the RandomHybrid algorithm

4 Experimental Results

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 planner [7]. All the instances represent plans of length 15. Each offer set of Instance I, 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 are as follows:

$$Q_{1,2} = \sum_{i=1}^{n} o_{j_{i},1}^{i}, \quad Q_{3,4} = \sum_{i=1}^{n} (o_{j_{i},1}^{i} + o_{j_{i},2}^{i}), \quad Q_{5,6} = \sum_{i=1}^{n} o_{j_{i},3}^{i}. \quad (5)$$
The set of the constraints of Instances from I to IV is defined as follows:

$$C_{1,2,3,4} = \{ (o_{ji,1}^i < o_{ji+1,2}^{i+1}) \mid i = 1, \ldots, n - 1 \}. \quad (6)$$

In the case of Instance V and VI, which are based on an example from [7], the constraints are the following:

$$C_{0,6} = \{ (o_{ji,1}^1 \leq 100), (o_{ji,2}^1 + o_{ji,2}^2 > 50), (o_{ji,5}^1 = o_{ji+1,1}^{i+1}) \mid i = 1, \ldots, n - 1 \}. \quad (7)$$

Besides the parameters introduced already in Section 3.3, 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 is 1000, the number of iterations is 100, the crossover probability is 95%, and the mutation probability is 0.5%. Moreover, all the experiments with the hybrid algorithms have been performed using $t = 20$, that is, the first SMT 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 1 to 3), and every experiment has been repeated 30 times on a standard PC with 2.8GHz CPU and Z3 [3] version 4.3 as SMT-solving engine.

The preliminary results of applying our hybrid algorithms to Instances I - VI are presented in Tables 1, 2, and 3, 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$^4$. 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 solutions 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.

$^4$ The pure GA-based planner has used the same parameters values as the hybrid ones. The test has been performed on the same machine.
Fig. 2. The comparison of the experimental results using two score functions.
### Table 1. Experimental results for Instances I and II

<table>
<thead>
<tr>
<th>exp</th>
<th>g</th>
<th>h</th>
<th>n</th>
<th>ind</th>
<th>int</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>10</td>
<td>9.9</td>
<td>13.0</td>
<td>15.7</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>8.2</td>
<td>13.1</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>4.1</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>22.4</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>22.4</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>76.8</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
<td>76.8</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>10</td>
<td>14.5</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>10</td>
<td>14.5</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>14.5</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
<td>14.5</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>10</td>
<td>14.5</td>
<td>13.0</td>
<td>13.0</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
<td>0.8</td>
<td>14.9</td>
</tr>
</tbody>
</table>

### Table 2. Experimental results for Instances III and IV

<table>
<thead>
<tr>
<th>exp</th>
<th>g</th>
<th>h</th>
<th>n</th>
<th>ind</th>
<th>int</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>10</td>
<td>13.0</td>
<td>27.6</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>10</td>
<td>12.4</td>
<td>25.4</td>
<td>16.7</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
<td>0.8</td>
<td>22.1</td>
</tr>
</tbody>
</table>

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

\[
score_{c}(P, t, Q) = P \cdot (Q - const), \quad score_{\ell}(P, t, Q) = \frac{P \cdot Q}{t},
\]

where \(P\), \(t\), and \(Q\) stand for the probability, the computation time, and the average quality, respectively, and \(const\)\(^6\) is a parameter, which value is selected

\(^6\) The values of the \(const\) parameters used for comparing the results for Instances I-VI are as follows: 1150, 1295, 2001, 1906, 386, 514, respectively.
Table 3. Experimental results for Instances V and VI

<table>
<thead>
<tr>
<th>exp</th>
<th>gn</th>
<th>ind</th>
<th>Int</th>
<th>Random</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>Semi-Random</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>Random</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
<th>Semi-Random</th>
<th>t[s]</th>
<th>Q</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>10</td>
<td>12.0</td>
<td>560.0</td>
<td>36.7</td>
<td>10.8</td>
<td>346.1</td>
<td>68.7</td>
<td>35.4</td>
<td>68.4</td>
<td>584.1</td>
<td>63.3</td>
<td>20.5</td>
<td>572.1</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>8.8</td>
<td>488.0</td>
<td>15.7</td>
<td>10.8</td>
<td>509.8</td>
<td>66.7</td>
<td>18.4</td>
<td>554.3</td>
<td>90.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>15.0</td>
<td>705.1</td>
<td>93.3</td>
<td>34.1</td>
<td>648.0</td>
<td>96.7</td>
<td>156.1</td>
<td>804.8</td>
<td>81.9</td>
<td>369.9</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td>20</td>
<td>36.2</td>
<td>638.2</td>
<td>80.0</td>
<td>22.7</td>
<td>556.1</td>
<td>92.3</td>
<td>96.8</td>
<td>722.8</td>
<td>95.3</td>
<td>40.6</td>
<td>660.1</td>
<td>96.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>14.9</td>
<td>777.9</td>
<td>96.7</td>
<td>51.8</td>
<td>697.1</td>
<td>96.7</td>
<td>298.7</td>
<td>888.1</td>
<td>131.8</td>
<td>783.0</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>20.0</td>
<td>667.3</td>
<td>83.3</td>
<td>28.2</td>
<td>684.1</td>
<td>100</td>
<td>165.9</td>
<td>808.9</td>
<td>81.5</td>
<td>361.4</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>16.9</td>
<td>620.8</td>
<td>96.7</td>
<td>31.8</td>
<td>569.8</td>
<td>100</td>
<td>124.4</td>
<td>660.3</td>
<td>42.2</td>
<td>546.8</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>20</td>
<td>20.1</td>
<td>561.1</td>
<td>93.3</td>
<td>23.0</td>
<td>532.6</td>
<td>98.7</td>
<td>75.1</td>
<td>608.8</td>
<td>90.0</td>
<td>42.5</td>
<td>588.0</td>
<td>96.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>42.0</td>
<td>829.8</td>
<td>100</td>
<td>139.9</td>
<td>731.4</td>
<td>100</td>
<td>1386.4</td>
<td>925.5</td>
<td>100</td>
<td>254.7</td>
<td>769.0</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>20.6</td>
<td>774.3</td>
<td>100</td>
<td>66.5</td>
<td>620.8</td>
<td>100</td>
<td>61.9</td>
<td>814.4</td>
<td>100</td>
<td>122.3</td>
<td>628.9</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>1</td>
<td>10</td>
<td>807.4</td>
<td>935.5</td>
<td>100</td>
<td>278.7</td>
<td>832.7</td>
<td>100</td>
<td>2614.5</td>
<td>993.3</td>
<td>100</td>
<td>583.2</td>
<td>874.4</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>20.4</td>
<td>852.6</td>
<td>100</td>
<td>260.1</td>
<td>778.7</td>
<td>100</td>
<td>1164.6</td>
<td>927.5</td>
<td>100</td>
<td>260.1</td>
<td>760.9</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for each Instance i from I to VI, to make the scores of the pure GA- and SMT-based algorithm equal. These scores are then selected as the benchmarks for the comparison given in Fig. 2. The dark- and light-grey bars correspond to the results obtained with the RH and SRH algorithm, respectively.

The score 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 score2 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 1, 2, and 3, as well as the highest chart bars in Figure 2 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.

5 Conclusions and Future Work

In this paper two prototypes 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 the method proposed is of a high potential. Our plan is 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. Moreover, using the experience gained from the concrete planning, we intend to develop also a hybrid solution for the abstract planning stage.

References