Evolutionary, symbolic, and hybrid algorithms for planning and web-service composition

Artur Niewiadomski
Siedlce University, Poland

Institute of Computer Science, Polish Academy of Sciences, 02.07.2020

Outline

PlanICS Team

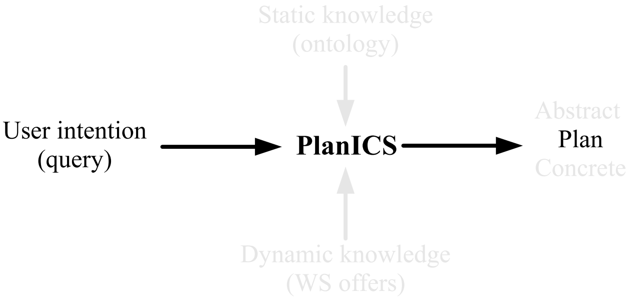
Intelligent hybrid system for planning and composition of web services

- Wojciech Penczek (ICS PAS, Warsaw)
  the Head of the project
- Artur Niewiadomski (Siedlce University)
  symbolic (SMT-based) computations and algorithms
- Piotr Switalski, Jaroslaw Skaruz (Siedlce University)
  Evolutionary (and other nature-inspired) Algorithms
- Mariusz Jarocki, Agata Polrola (Lodz University)
  main concepts and PlanICS language contributors
- Lukasz Mikulski (Nicolaus Copernicus University, Torun)
  Multiset Explorer - plan linearisations
- Maciej Szreter (ICS PAS, Warsaw)
  dynamic Web services

Related work - Web Service Composition Systems

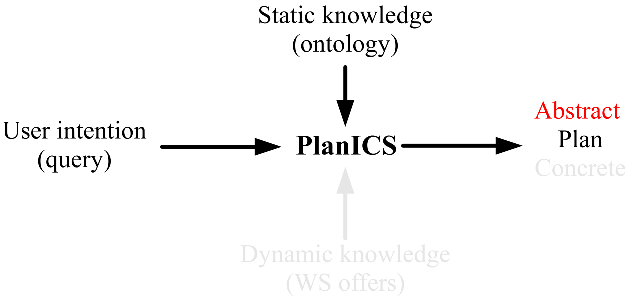
- Entish - IOPR, a two phase planning by an ontology,
- WSMO - ontology, IOPR, a formal goal, embedded rule languages
- WSMX - WSMO implementation, service registration, service discovery by matchmaking, service activation by adapters
- SUPER - composition based on WSMO ontology and AI algorithms
- PlanICS
  - a state-based approach, multi-phase planning, a simple rule language,
  - abstract planners based on GA, SMT-solvers, and combining both as **hybrid planners**,
  - concrete planners based on evolutionary algorithms, SMT-solvers, and hybrid ones.

Key Concepts



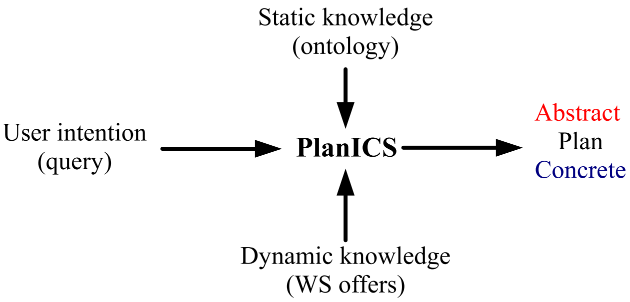
- The main goal: an arrangement of service executions satisfying a user intention

Key Concepts



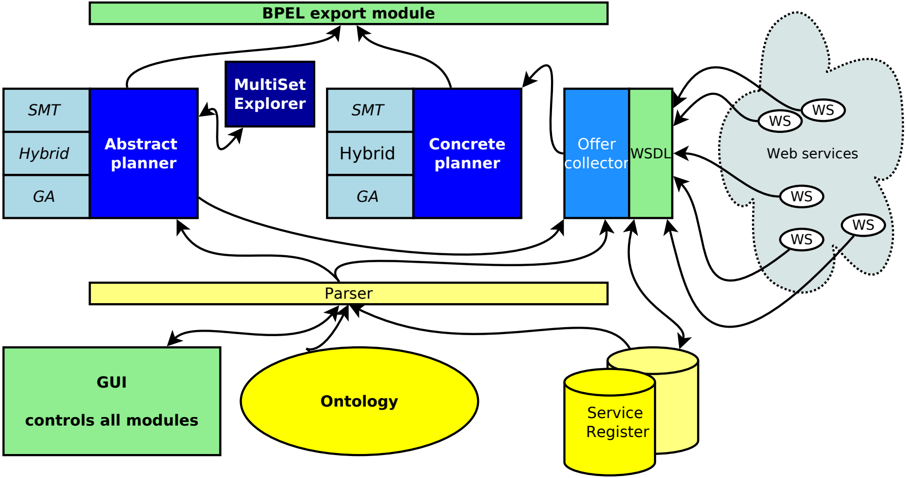
- The main goal: an arrangement of service executions satisfying a user intention
- Ontology - the types of services and objects

Key Concepts

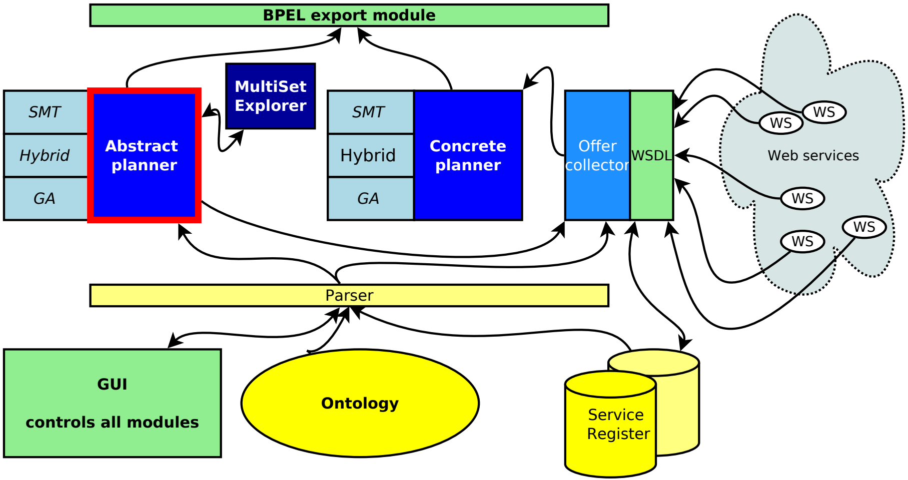


- The main goal: an arrangement of service executions satisfying a user intention
- Ontology - the types of services and objects
- A two phase composition process: abstract (on types) and concrete (on web services)

System Overview



System Overview



Abstract Planning Phase

Planning in the terms of

- Service types
- Object types
- Abstract values of object attributes

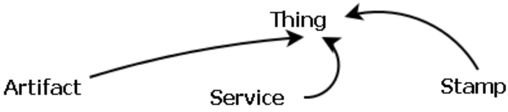
Basic concepts

- A **world** - a set of objects with specific attribute values
- A **service** can **transform** a world (if the pre-condition is met) by changing attribute values of existing objects and adding new objects
- A user **query** specifies **initial** and **expected** (final) worlds
- A **solution** is a **sequence** of **service types** able to transform an initial world into a world matching an expected one
- A **plan** is a set of solutions built over the same **multiset** of service types, regardless the ordering and the contexts.

Main goals of abstract planning

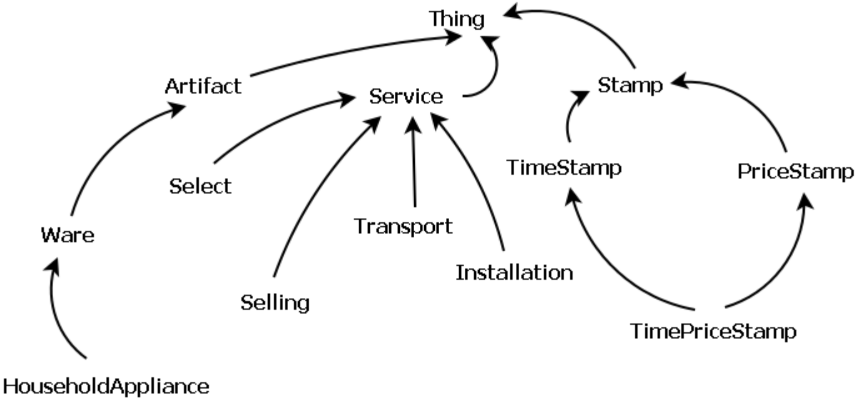
- Checking whether the user query can be realized using a given ontology
- Reducing the search space for a concrete planner
- Reducing the number of network interactions between web services and offer collector
- Providing a number of different potential ways to realize the query

Ontology



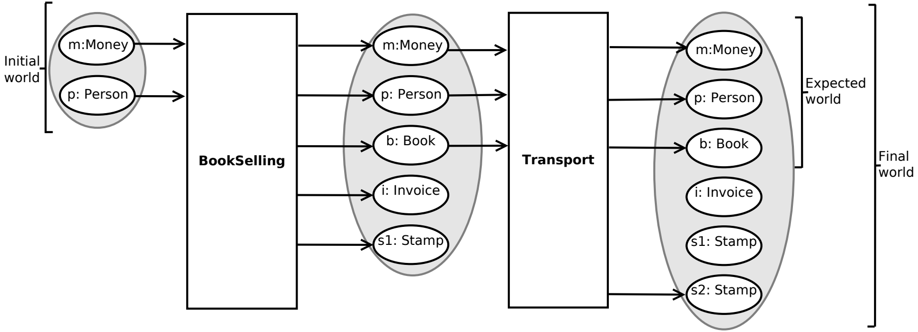
- OWL + embedded PlanICS language
- Service types
- Artifacts - objects the services operate on
- Stamps - special objects describing certain execution features

Ontology Example



User Query and Abstract Plan Example

in $\{p : Person\}$
inout $\{m : Money\}$
out $\{b : Book\}$
pre $(p.name = ME \ and \ p.address = MyAddr. \ and \ m.amount = 50)$
post $(b.title = \text{"}Java \ in \ Practice\text{"} \ and \ b.location = p.address)$



SMT-based abstract planning

$$\varphi_k^q = \mathcal{I}^q \bigwedge_{i=1..k} \left(\mathcal{C}_i \bigvee_{s \in \mathbb{S}} \mathcal{T}_i^s \right) \wedge \mathcal{E}_k^q \wedge \mathcal{B}_k^q$$

- Abstract planning problem for a query q encoded as the formula φ_k^q
- φ_k^q satisfiable iff there exists a solution for q of the length k
- If a solution is found, then block all known abstract plans with the formula $\mathcal{B}_k^q$ and search for other solutions,
- Otherwise proceed with $k + 1$

SMT-based abstract planning

$$\varphi_k^q = \mathcal{I}^q \bigwedge_{i=1..k} \left(\mathcal{C}_i \bigvee_{s \in \mathbb{S}} \mathcal{T}_i^s\right) \wedge \mathcal{E}_k^q \wedge \mathcal{B}_k^q$$

- Abstract planning problem for a query q encoded as the formula φ_k^q
- φ_k^q satisfiable iff there exists a solution for q of the length k
- If a solution is found, then block all known abstract plans with the formula $\mathcal{B}_k^q$ and search for other solutions,
- Otherwise proceed with $k+1$

The formula φ_k^q encodes

- the initial worlds
- contexts and worlds transformations
- the expected worlds
- a blocking formula

SMT-based abstract planning

$$\varphi_k^q = \mathcal{I}^q \bigwedge_{i=1..k} \left(\mathcal{C}_i \bigvee_{s \in \mathbb{S}} \mathcal{T}_i^s \right) \wedge \mathcal{E}_k^q \wedge \mathcal{B}_k^q$$

- Abstract planning problem for a query q encoded as the formula φ_k^q
- φ_k^q satisfiable iff there exists a solution for q of the length k
- If a solution is found, then block all known abstract plans with the formula $\mathcal{B}_k^q$ and search for other solutions,
- Otherwise proceed with $k + 1$

The formula φ_k^q encodes

- the initial worlds
- contexts and worlds transformations
- the expected worlds
- a blocking formula

SMT-based abstract planning

$$\varphi_k^q = \mathcal{I}^q \bigwedge_{i=1..k} \left(\mathcal{C}_i \bigvee_{s \in \mathbb{S}} \mathcal{T}_i^s \right) \wedge \mathcal{E}_k^q \wedge \mathcal{B}_k^q$$

- Abstract planning problem for a query q encoded as the formula φ_k^q
- φ_k^q satisfiable iff there exists a solution for q of the length k
- If a solution is found, then block all known abstract plans with the formula $\mathcal{B}_k^q$ and search for other solutions,
- Otherwise proceed with $k+1$

The formula φ_k^q encodes

- the initial worlds
- contexts and worlds transformations
- the expected worlds
- a blocking formula

SMT-based abstract planning

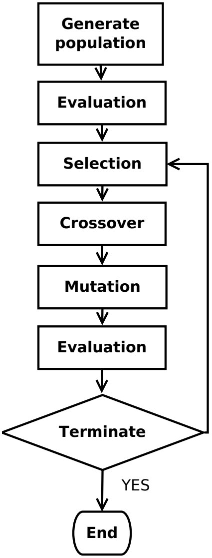
$$\varphi_k^q = \mathcal{I}^q \bigwedge_{i=1..k} \left(\mathcal{C}_i \bigvee_{s \in \mathbb{S}} \mathcal{T}_i^s \right) \wedge \mathcal{E}_k^q \wedge \mathcal{B}_k^q$$

- Abstract planning problem for a query q encoded as the formula φ_k^q
- φ_k^q satisfiable iff there exists a solution for q of the length k
- If a solution is found, then block all known abstract plans with the formula $\mathcal{B}_k^q$ and search for other solutions,
- Otherwise proceed with $k+1$

The formula φ_k^q encodes

- the initial worlds
- contexts and worlds transformations
- the expected worlds
- a blocking formula

Genetic Algorithm



- Introduced in 1960 by John Holland
- Applied to optimization and search problems
- A population of individuals (candidate solutions) is evolved toward better solutions
- Operators: mutation, crossover and selection
- Problem specific:
  - Encoding of individuals
  - Fitness function
  - Versions of operators and probabilities of their application

An individual is represented usually by a fixed-length array of bits or numbers. The fitness function evaluates a candidate solution - we know which ones are better than others. The better individuals have more chances to move on to the next stages of the algorithm.

GA-based abstract planning

- Standard GA implementation, but sophisticated fitness function and specialised mutation operator
- Individual: a sequence of service types
- Genes reordering in order to find the longest executable prefix
- **Good service type** concept

Intuitively, a service type is good, if it produces objects that can be a part of the expected world, or they can be an input for other good service types.

Individual of GA

An individual

- a multiset M of service types.

Abstract plan

Selling → Payment → Transport

Individual of GA

1	6	3

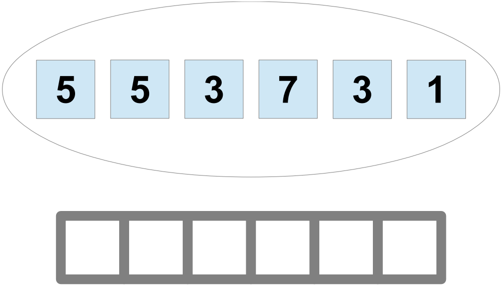
Set of all services and their indexes

Index	Service
1	Selling
…	…
3	Transport
…	…
…	…
6	Payment

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.

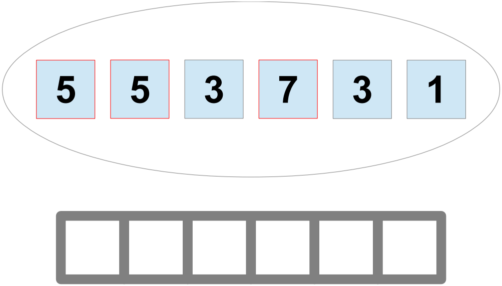


- Start from an empty sequence

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.

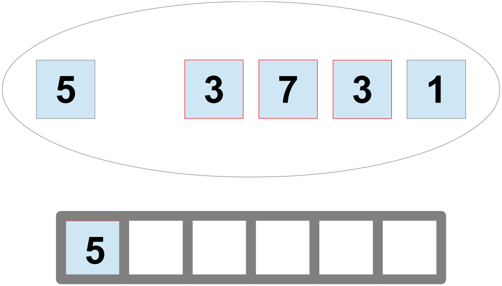


- Service types able to transform an initial world

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.

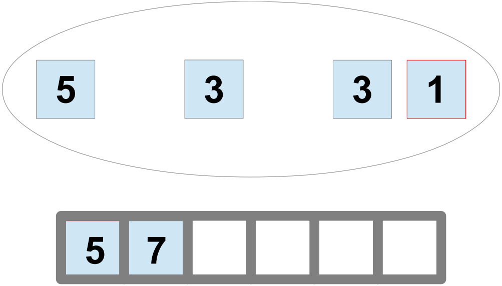


- Choose one, append it to seq_M ...

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.

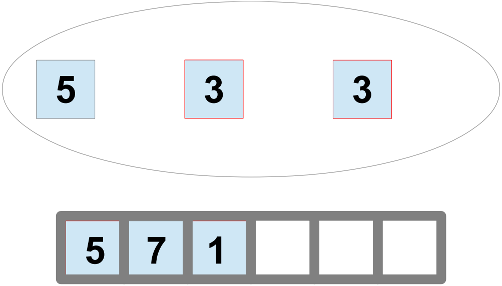


- ...and check which Service Type can transform the current world

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.

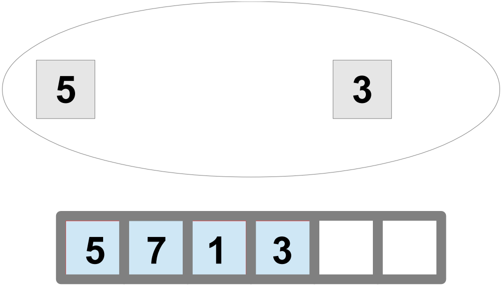


- ...append and check next ...

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.

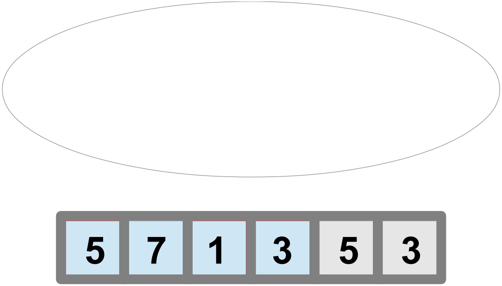


- If none of remaining Service Types can transform the current world

Sequence from Multiset

Checking whether M is a plan

A transformation sequence seq_M is constructed from M.



- Append all to seq_M

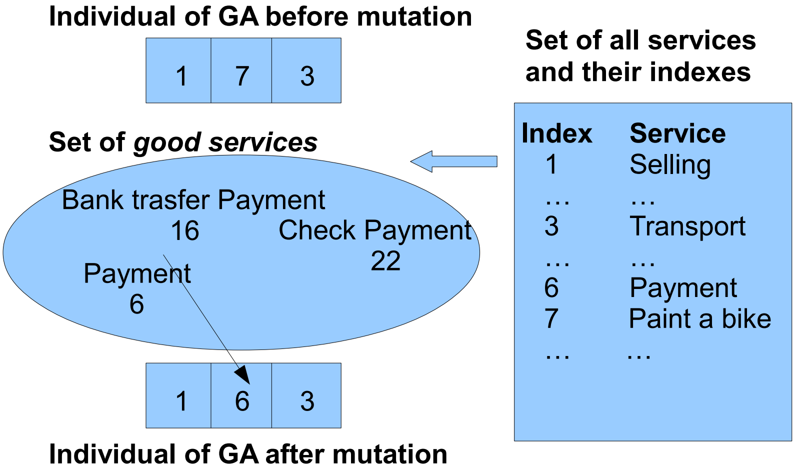
Fitness function

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

- f_{wM} is the maximal number of objects from w_M, which **types and valuations** are consistent with objects from an expected world,
- $c_{wM} = \min(\text{cst}(w_M), |w_q|))$, where $\text{cst}(w_M)$ is the number of the objects from w_M of **types** consistent with an expected world
- g_{seqM} is the number of the good service types occurring in seq_M,
- l_M is the length of the executable prefix of seq_M
- $\alpha = 0.1$, $\beta = 0.7$, $\gamma = 0.1$, and $\delta = 0.2$ are parameters of the fitness function.

After finding the first solution, the fitness function is modified by a penalty for similarity of new solutions to those already found.

Mutation operator



Hybrid Solution

Motivation

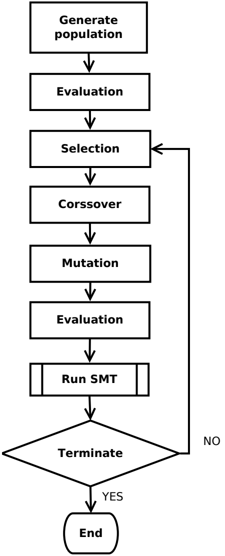
- Advantages and disadvantages of both methods: SMT and GA

	SMT	GA
Short time	✗	✓
High probability	✓	✗

Solution

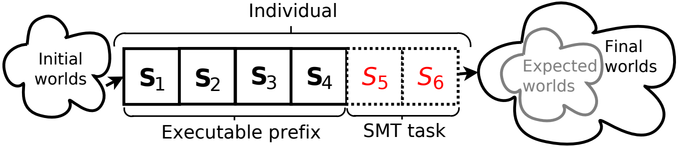
- Combine both algorithms to exploit their advantages

Main Concept of Hybrid Abstract Planner



SMT-based procedure

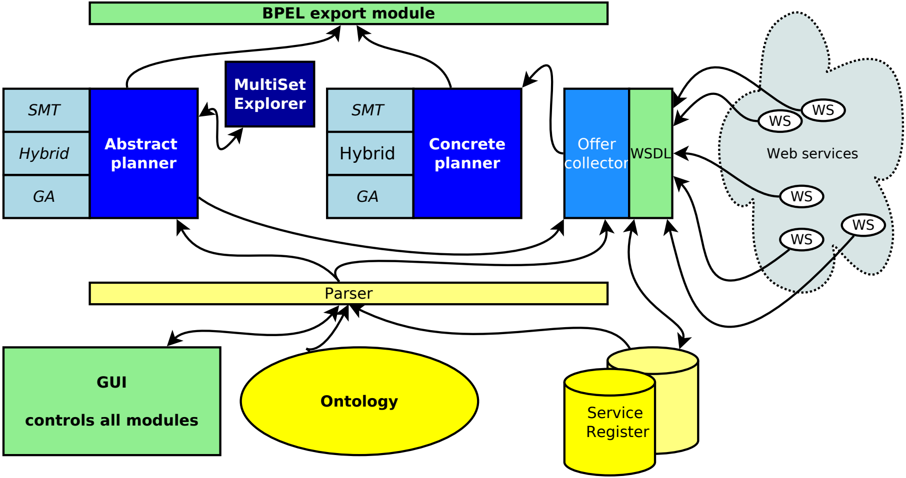
- searching for a new suffix of an individual
- the individuals passed to SMT have to
  - consist at least in a half of good service types, and
  - at least a half of their genes should constitute an executable prefix.



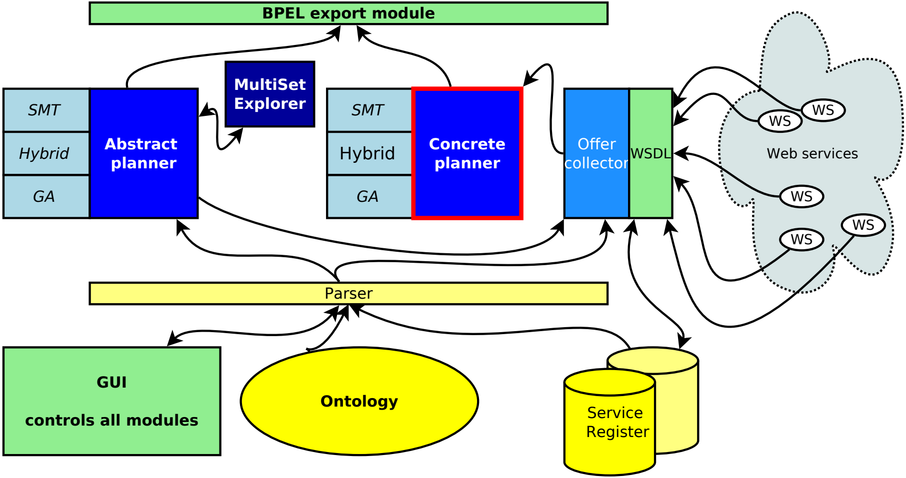
Experimental results

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

System Overview

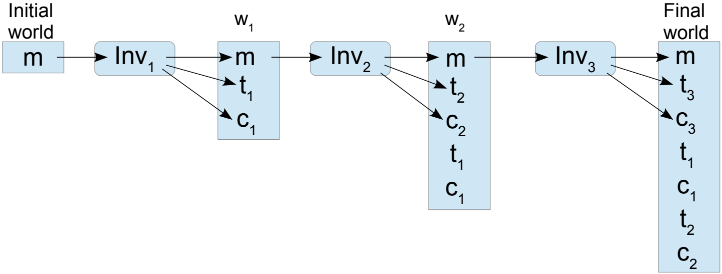


System Overview



Abstract Plan, Offers, Constraints

Query: 3 investments up to \$100, maximizing the sum of profits



$$\begin{bmatrix} o^1_{1,1} & \dots & o^1_{1,5} \\ \vdots & \ddots & \vdots \\ o^1_{k_1,1} & \dots & o^1_{k_1,5} \end{bmatrix} \begin{bmatrix} o^2_{1,1} & \dots & o^2_{1,5} \\ \vdots & \ddots & \vdots \\ o^2_{k_2,1} & \dots & o^2_{k_2,5} \end{bmatrix} \begin{bmatrix} o^3_{1,1} & \dots & o^3_{1,5} \\ \vdots & \ddots & \vdots \\ o^3_{k_3,1} & \dots & o^3_{k_3,5} \end{bmatrix}$$

Attributes:	$m.amount$	$t_i.amount$	$t_i.profit$	$c_i.fee$	$m.amount'$
Variables:	$o^i_{j,1}$	$o^i_{j,2}$	$o^i_{j,3}$	$o^i_{j,4}$	$o^i_{j,5}$

Constraint: $\sum_{i=1}^{3} \ o^i_{j,2} \leq 100$, **Quality function:** $\sum_{i=1}^{3} \ o^i_{j,3}$

Concrete Planning as the Constrained Optimization Problem

$P^i_j = [o^i_{j,1}, o^i_{j,2}, \ldots, o^i_{j,m_i}]$

$\mathbb{P}$: the set of all possible sequences $(P^1_{j_1}, \ldots, P^n_{j_n})$,

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

- $Q : \mathbb{P} \mapsto \mathbb{R}$,
  an objective function defined as the sum of all quality constraints
- $\mathbb{C}(S)$, where $S \in \mathbb{P}$,
  a set of constraints to be satisfied.

A solution of CPP

- selecting one offer from each offer set
  - all constraints are satisfied
  - value of the objective function is maximized

SMT-based algorithm overview

SMT(Satisfiability Modulo Theories)

- Encode CPP as a formula $[\varphi]$ which is satisfiable iff there is a solution
- Exploit an SMT-solver to find a solution of quality $\boldsymbol{q}$, where $q \in \mathbb{R}$
- Proceed with $[\varphi] \wedge [\boldsymbol{quality} > \boldsymbol{q}]$ to search for better solutions
  - adapting the binary search method
  - taking advantage of SMT interactive mode
  - using assumptions

GA for Concrete Planning Problem

Standard GA implementation

- Individual - a sequence of indices of the offers chosen from the consecutive offer sets,
- $fitness(Ind) = Q(S_{Ind}) + \beta \cdot \frac{|sat(\mathbb{C}(S_{Ind}))|}{c}$,
  - Ind - an individual,
  - S_{Ind} - a sequence of the offer values corresponding to Ind,
  - $sat(\mathbb{C}(S_{Ind}))$ - a set of the constraints satisfied by Ind,
  - c - the number of all constraints,
  - β - a constant to reduce both of the sum components to the same order of magnitude.

Simmulated Annealing (SA)

- a probabilistic metaheuristic
- the annealing process in metallurgy - heating and cooling of a material in order to improve its properties
- in SA "a material" is a potential solution - **an individual**
- cooling implemented as a slow decrease in the probability of accepting worse solutions while the algorithm explores the search space
- search space exploration by applying a neighbourhood operator to the individual in order to obtain a new individual
  - usually, a neighbourhood operator is like mutation in GA, but applied with the probability equals to 1
  - in our case it changes one randomly chosen "gene- an offer index

SA Algorithm for Concrete Planning

SA(n, G, I_L, $temp$, $decFactor$)
Input: the length of the individual: n, the number of iterations: G, the number of internal loop iterations: I_L, an initial value of the temperature: $temp$, the temperature decreasing factor: $decFactor$
Result: a solution of the highest quality function value

```
begin
    I_cur ← random(n) ;                                  // generate an individual randomly
    Q_cur ← Q(I_cur) ;                     // calculate the quality of the initial individual
    Q_best ← Q_cur ;                              // remember the best quality found so far
    I_best ← I_cur ;                                 // store the best solution found so far
    for (i ← 1..G) do
        for (j ← 1..I_L) do
            I_new ← neighbourhood(I_cur) ;                    // generate a new individual
            if (I_new satisfies all constraints) then
                Q_new ← Q(I_new) ;  // calculate the quality of the new individual
                if (Q_new > Q_cur) ∨ (random([0.0, 1.0]) < exp((Q_new − Q_cur)/temp)) then
                    I_cur ← I_new;
                    Q_cur ← Q_new;
                    if (Q_cur > Q_best) then
                        Q_best ← Q_cur ;    // update the best quality value found
                        I_best ← I_cur ;          // update the best solution - elite
        temp ← temp ∗ decFactor;                               // decrease the temperature
    return I_best;
```

Generalized Extremal Optimization (GEO)

Different terminology!

- Introduced by Sousa et al. about 2004
- The main idea is to focus on the worst parts of a solution and change them
- Similarly to SA maintains a single solution
- At every step, the algorithm
  - tries to mutate **every** gene separately
  - assigns a number proportional to the gain (or loss) of fitness after the change
  - builds a ranking of genes - the most promising have a better chance to mutate
  - stochastically choose and modify a gene from ranking

GEO Algorithm for Concrete Planning

GEO(n, K, τ)
Input: the number of individuals in population: n, the number of iterations: K, a parameter: τ
Result: a solution with the highest fitness value

```
begin
    I_cur ← random(n) ;                                   // generate an initial offer vector randomly
    Q_best ← Q(I_cur) ;                                   // calculate and store the best fitness value found so far
    I_best ← I_cur ;                                      // remember the best solution found so far
    for (i ← 1..K) do
        for (j ← 1..n) do
            I_tmp,j ← mutation(I_cur, j) ;                // mutate the j-th offer index
            if (I_tmp,j satisfies all constraints) then
                Q_j ← Q(I_tmp,j);                         // calculate the fitness value of I_tmp,j
            else
                Q_j ← ∞;                                  // individuals violating constraints fall low in the ranking
            Δ_j ← Q_j − Q(I_cur);                         // relative change of fitness resulting from mutation
        Rank ← sort((I_tmp,1, Δ_1), . . . , (I_tmp,n, Δ_n)), desc);     // build the ranking by sorting the
         mutated populations according to decreasing Δ_j values
        changed ← false ;
        while (¬changed) do
            j ← random(1..n) ;                            // randomly choose an individual to be changed
            k ← Rank.find(j) ;                            // the position of the j-th individual in the ranking
            p ← k^(−τ) ;                                  // the probability of mutation of the j-th individual
            x ← random([0.0, 1.0]);                       // a random value from the range [0.0, 1.0]
            if (p > x) then
                I_cur ← I_tmp,j ;                         // a new population becomes the current one
                changed ← true;
                if ((∞ > Q_j > Q_best)) then
                    Q_best ← Q_j ;                        // update the best fitness value found so far
                    I_best ← I_cur ;                      // update the best solution found so far
    return I_best;                                        // return the best solution
```

Hybrid Solution

Motivation

- Disadvantages of both methods: SMT and Metaheuristics

	SMT	Metaheuristics
Short time	✗	✓
Good quality	✓	✗
High probability	✓	✗

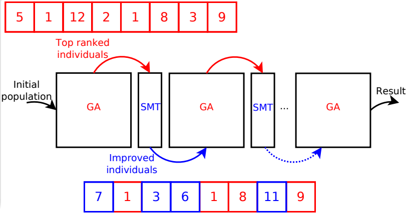
Solution

- Combine both algorithms to exploit their advantages

Hybrid Concrete Planners

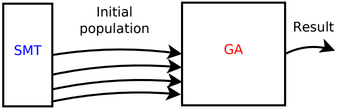
Random Hybrid (RH) and Semi-Random Hybrid (SRH)

- combine GA with SMT
- alternately run GA and SMT
- best individuals of GA passed to SMT for improving



Initial Population Hybrid (IPH)

- SMT generates (a part of) the initial population
- the generated individuals satisfy all constraints
- GA obtains some (usually not optimal) solutions at start



Hybrid SA and Hybrid GEO

HSA and HGEO

- follows the IPH scheme
- GA replaced by SA or GEO
- the initial solution generated by an SMT procedure
- it satisfies all constraints, but usually is of poor quality

Experiments

- 12 datasets generated randomly, scaled by the length of a plan (10, 15, or 20) and the number of offers ($2^8 = 256$ or $2^9 = 512$)
- the search space varying from $256^{10} = 2^{80}$ to $512^{20} = 2^{180}$

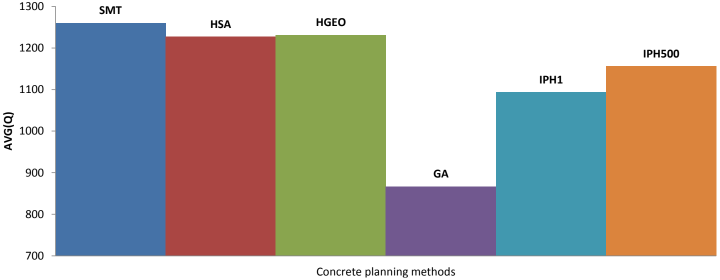
$$\mathbb{C}(S)_{1..6} = \bigwedge_{i=1}^{n-1} (o^i_{j_i,1} < o^i_{j_{i+1},1}), \qquad Q_{1..6} = \sum_{i=1}^{n} o^i_{j_i,2}$$

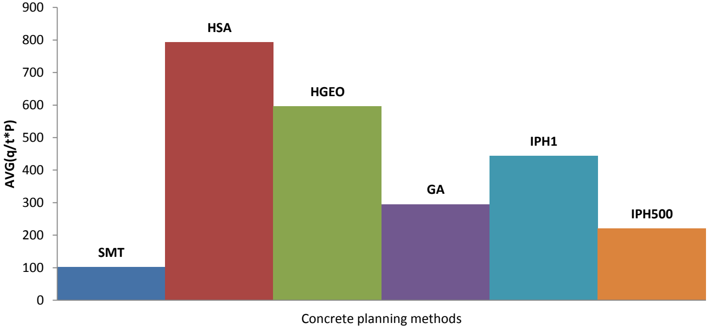
$$\mathbb{C}(S)_{7..12} = \bigwedge_{i=1}^{n-1} (o^i_{j_i,1} - o^{i+1}_{j_i,2}) > 10, \quad Q_{7..12} = \sum_{i=1}^{n-1} (o^i_{j_i,1} - o^{i+1}_{j_i,2}),$$

Parameters

- IPH and GA: the population size = 1000, iterations = 100, crossover probability 95%, mutation probability 0,5%,
- HSA: $temp = 1.0$, $G = 500$, $I_L = 40$, $decFactor = 0.98$,
- HGEO: $n = 10; 15; 20$, $K = 1000$, $\tau = 5.0$,
- SMT: 400 sec. timeout.

Experimental Results Summary





Conclusions and Future Work

Hybrids are

- methods of a high potential
- a trade-off between time and probability

Future work

- Improve the results of APP and CPP
- Try to combine SMT with other Evolutionary Algorithms
- Develop hybrids for other problems, like, e.g., model checking
- Reductions and abstractions to speed up the planning

Thank You

References

- Combining ontology reductions with new approaches to automated abstract planning of PlanICS. Applied Soft Computing 53 (2017): 352-379, http://dx.doi.org/10.1016/j.asoc.2017.01.007
- Concrete Planning in PlanICS Framework by Combining SMT with GEO and Simulated Annealing. Fundam. Inform. 147 (2016): 289-313, IOS Press, 2016. DOI 10.3233/FI-2016-1409
- Hybrid Approach to Abstract Planning of Web Services. Service Computation 2015 : 35-40
- Genetic Algorithm to the Power of SMT: a Hybrid Approach to Web Service Composition Problem. Service Computation 2014: 44-48
- Towards SMT-based Abstract Planning in PlanICS Ontology, in KEOD, pages 123-131, 2013
- Automated abstract planning with use of genetic algorithms, GECCO (Companion) 2013: 129-130
- Evolutionary Algorithms for Abstract Planning, in PPAM (1), vol. 8384 of LNCS, pages 392-401, Springer, 2013
- SMT vs Genetic Algorithms: Concrete Planning in PlanICS Framework, CS&P2013: 309-321, `ceur-ws.org/Vol-1032/paper-27.pdf`