Alternative Approaches to Planning

Otakar Trunda
Department of Theoretical Computer Science and Mathematical Logic,
Charles University in Prague, Prague, Czech Republic

1 INTRODUCTION

In my PhD. dissertation, I focus on action planning and constrained discrete optimization. I try to introduce novel approaches to the field of single-agent planning by combining standard techniques with *meta-heuristic* optimization, *machine-learning* algorithms, *hyper-heuristics* and *algorithm selection* approaches.

Our main goal is to create new and flexible planning algorithms which would be suited for a large variety of real-life problems. Planning is a fundamental and difficult problem in AI and any new results in this area are directly applicable to many other fields. They can be used for single-agent or multi-agent action selection in both competitive or cooperative environment and as we focus on optimization, our techniques are suitable for real-life problems that arise in robotics or transportation.

2 BACKGROUND

In this section, we provide a brief description of notions and research topics that we refer to later in the paper.

2.1 Planning

Planning deals with problems of selection and causally ordering of actions to achieve a given goal from a known initial situation. Planning algorithms assume a description of possible actions and attributes of the world states in some modelling language such as Planning Domain Description Language (PDDL) as its input. This makes the planning algorithms general and applicable to any planning problem starting from building blocks to towers and finishing with planning transport of goods between warehouses (Ghallab et al., 2004).

A state which satisfies the goal condition is called a *goal state*, a sequence of actions $(a_1, ..., a_n)$ is called a *plan*, if executing these actions one by one starting in the initial state leads to some goal state.

There are two different kinds of planning tasks - in the *satisficing planning*, we are interested in finding just any *plan*, while in the *optimization planning* we want to find a *plan* which minimizes given objective function.

In the *satisficing planning*, however, we still consider some solutions to be better than others, we prefer shorter plans. An example of a satisficing planning task might be a *Sokoban* problem or a *Rubik's cube*. Finding even suboptimal solutions in these domains is difficult (at least for large instances).

The typical representatives of an optimization planning are transportation problems, where the task is to deliver some goods to specific locations and minimize the time requirement and fuel consumption.

2.2 Meta-heuristics

Meta-heuristics (or Modern heuristics) are optimization algorithms that don't guarantee finding optimal solutions, but can often find high-quality solutions with reasonable search effort (Rothlauf, 2011). Examples of popular meta-heuristics are Genetic Algorithms, Particle Swarm Optimization, Ant Colony Optimization, Simulated Annealing, and others.

These techniques can be divided into two groups according to the type of search space. Particle Swarm Optimization or Simulated Annealing usually work on continuous space, while Ant Colony Optimization works on discrete space. Genetic Algorithms are very general and are able to work on both continuous and discrete search spaces.

The search space of planning problems is inherently discrete, so we are interested in algorithms that can handle discrete structures. Commonly used structures are graphs, set of permutations, or in case of planning - set of finite sequences of actions.

Trunda O..
Alternative Approaches to Planning.

2.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a stochastic optimization algorithm that combines classical tree search with random sampling of the search space. The algorithm was originally used in the field of game playing where it became very popular, especially for games Go and Hex. A single player variant has been developed by Schadd et al. (Schadd et al., 2008) which is designed specifically for single-player games and can also be applied to optimization problems. The MCTS algorithm successively builds an asymmetric tree to represent the search space by repeatedly performing the following four steps:

1. *Selection* – The tree built so far is traversed from the root to a leaf using some criterion (called *tree policy*) to select the most urgent leaf.
2. *Expansion* – All applicable actions for the selected leaf node are applied and the resulting states are added to the tree as successors of the selected node (sometimes different strategies are used).
3. *Simulation* – A pseudo-random simulation is run from the selected node until some final state is reached (a state that has no successors). During the simulation the actions are selected by a *simulation policy*,
4. *Update/Back-propagation* – The result of the simulation is propagated back in the tree from the selected node to the root and statistics of the nodes on this path are updated according to the result.

The core schema of MCTS is shown at Figure 1 from (Chaslot et al., 2008).

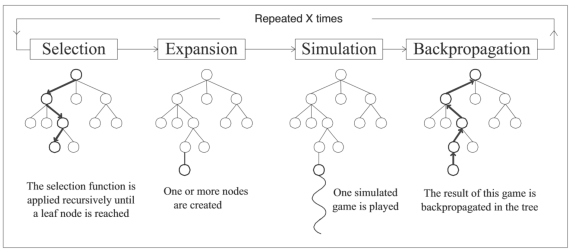

Figure 1: Basic schema of MCTS (Chaslot et al., 2008).

One of the most important parts of the algorithm is the *node selection criterion* (a tree policy). It determines which node will be expanded and therefore it affects the shape of the search tree. The purpose of the tree policy is to solve the exploration vs. exploitation dilemma.

Commonly used policies are based on a so called *bandit problem* and *Upper Confidence Bounds for Trees* (Auer et al., 2002; Kocsis and Szepesvári, 2006) which provide a theoretical background to measure quality of policies. We use standard tree policy for the single-player variant of MCTS (SP-MCTS) due to Schadd et al. (Schadd et al., 2008) that is appropriate for planning problems (planning can be seen as a single-player game where moves correspond to action selection).

The behaviour of MCTS can be seen on an example in figure 2. In the yellow field there is a function to be minimized and above it there is a tree build by MCTS algorithm. Function values are used as results of the simulations. We can see that the algorithm identifies promising regions and focuses the sampling on these regions.

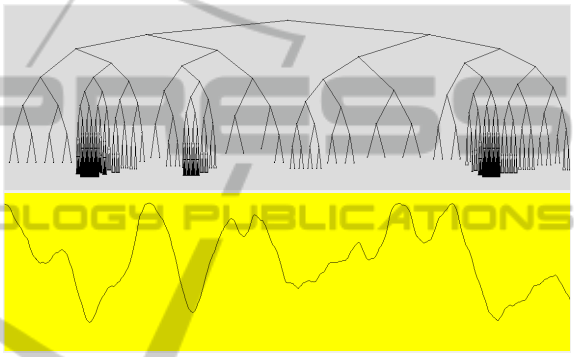

Figure 2: Simple example of MCTS tree.

2.4 Algorithm Selection

Algorithm selection is a relatively new field which deals with the problem of selecting the right algorithm for specified task. Currently, the research in this area focuses mostly on selecting classification algorithms.

It has also been used to select search algorithms for SAT instances, but so far not for planning.

The algorithm selection problem is closely related to *hyper-heuristics* and *parameter tuning*. Hyper-heuristics search for the best search algorithm for given problem instance by combining so called *low level heuristics*. The motivation behind this idea is that the human expert uses only intuition when designing for example mutation and recombination operators. Such intuition may or may not be correct and furthermore different instances may require different setting. Hyper-heuristics search the configuration space automatically and often can find better solution.

3 MOTIVATION

Currently, the most efficient approach to solve planning problems is heuristic forward search (mostly

in the form of *A** or *Hill-climbing*). In the paper (Toropila et al., 2012), we showed that classical planners are not competitive when solving a real-life transportation planning problem of the Petrobras company (Vaquero et al., 2012). The paper proposed an ad-hoc Monte Carlo Tree Search (MCTS) algorithm that beat the winning classical planner SGPlan in terms of problems solved and solution quality.

We believe, that there are many more planning domains where the classical planners wouldn't perform well and different techniques are needed. Reason for this is that newly developed planners are tested on domains from the *International Planning Competition (IPC)* (Olaya et al., 2014) which focuses on artificial problems.

There are two main tracks on the *IPC*

- in the *satisficing track*, the goal is to find *any* plan, but the solution quality is considered in the evaluation. In this track, quite hard problems are used for the evaluation, problems where just finding any plan is difficult.
- in the *optimal track*, the task is to find either a *length-optimal* plan (plan with the minimal number of actions) or a *cost-optimal* plan which minimizes the sum of actions' costs. In this track, only optimal solutions are accepted.

Real-life problems, however, don't really fit in any of those categories. Unlike the *satisficing track*, finding *any* solution is usually easy in practical problems and the optimization part is the real issue, but unlike in the *optimal track*, we might not want to guarantee optimality, since it would take too much time.

Consider the following example: the *Travelling Salesman Problem (TSP)* can be viewed as a planning problem, where in the initial state the agent is located in some virtual location from which it can move to any city using zero-cost action, then it can travel between the cities and a goal state is reached when all the cities have been visited.

Many different techniques has been devised in the past to solve this problem, most of them are based on meta-heuristics, which don't guarantee finding the optimal solution, but can find high-quality solutions in a reasonable time. Classical planner, however, would use an *A** algorithm, which is not well suited for this problem. (Although this example seems far-fetched, practically motivated planning domains often involve some kind of transportation therefore optimization planning might be close to solving some constrained version of TSP.)

On the other hand, to optimally solve the previously mentioned *Rubik's Cube* problem, the *A** algorithm is a good choice and classical planners would perform well on this domain.

Furthermore, we have no a priori knowledge about the shape of the fitness landscape of the problem. As the PDDL language is very general, it can describe all kinds of problems. It can be compared to *General Game Playing* (GGP) (Genesereth et al., 2005) - a logic-based formalism to describe rules of combinatorial games like *chess*, *go* and others.

Several GGP algorithms exist which take description of a game is their input and are able to play any game that can be described in GGP. Such GGP players, however, are far less efficient than engines specialized to just one game (like chess engines). Domain specific engines often use very different algorithms, for example *chess* engines are based on the alpha-beta algorithm while the best *Go* engines use MCTS.

Obviously, different problems require different techniques so it is important to select the proper search algorithm.

To sum up, the current techniques use mostly *A** and its variants and focus on satisficing problem rather than optimization. Those techniques are quite rigid as they use the same algorithm on all domains. Research in this area focuses mostly on developing new heuristic estimators for *A**.

4 STAGE OF THE RESEARCH

In the current stage of our research, we have studied published approaches and related work and we have already identified several opportunities where our approach could improve standard techniques. We have implemented some of those ideas and published the results. Since I am currently in the second year of my PhD. study, I will continue this research at least for the next two years.

The results we published so far cover these topics

- Solving real-life logistic problem *Petrobras* by MCTS (Toropila et al., 2012)
- Generalizing the MCTS approach to solve any logistic domain (Trunda and Barták, 2013)
- Improving the Red-Black planning heuristic by machine learning (Trunda and Barták, 2014)
- Automatic creation of pattern databases by meta-heuristics (Trunda, 2014)

These results deal with specific problems which in our further work we would like to extend and generalize to wider range of domains.

5 OUTLINE OF OBJECTIVES

The main objective of our research is to combine standard planning algorithms with optimization meta-heuristics and other techniques of soft-computing. There are three fundamental ways to do that

1. *use meta-heuristics as a preprocessing to improve the performance of standard techniques.*

   We have already published two papers that fall into this category (Trunda and Barták, 2014; Trunda, 2014) and we believe that there are many more opportunities to improve standard planners in this manner.

   For example, a *Symbolic search* algorithm works with *Binary decision diagrams (BDDs)* and the efficiency of this data structure is highly dependent on the ordering of variables (which is problem-dependent). Finding some good ordering before the actual search is a typical example of an optimization preprocessing.

   We believe that such optimizations are important in order to make the planning system flexible, robust and efficient.

2. *use meta-heuristics to solve the planning problems directly*

   We already have some experience with using MCTS for planning (Toropila et al., 2012; Trunda, 2013; Trunda and Barták, 2013). We would like to work further in this area and also find other techniques that could be used directly for optimization planning.

   This research should shed some light on the problem of which features of planning domains are important in order to select the proper search algorithm.

   Since some domains are inherently not suited for the use of meta-heuristics, an important part of this task will be to find transformations or reformulations of the problem which would make it more suitable to optimization algorithms.

3. *use meta-heuristics or machine-learning to devise an* algorithm selection *technique for planning.*

   We will analyse this issue later.

6 RESEARCH PROBLEM

In this section, we address possible problems with accomplishing the research objectives.

With using meta-heuristics as a preprocessing to standard techniques, there is an important issue of distributing the computation time. Let t be a problem instance, by $time(t,h)$ we denote the time required to solve the problem t, where h is some information that can help us (like what algorithm to use or how to configure it). In the preprocessing phase, we try to find h which will help us the most. Time to find h we denote by $find(h)$.

In order for the preprocessing to have any positive effect, equation 1 has to hold.

$$find(h) + time(t,h) \leq time(t,NoHelp) \quad (1)$$

Techniques used for the preprocessing are usually *anytime*, which means that if we let then run longer, we might get a better solution. Such better solution h would lead to smaller $time(t,h)$, but if we allocate to much time for the preprocessing phase, it may not pay off as the equation 1 might not hold. Furthermore, we don't know a priori the value $time(t,NoHelp)$ and it is not easy to deduce $time(t,h)$ either.

Another problem rises with the need of an evaluation function during the preprocessing phase. Meta-heuristics work with a population of solutions and use an evaluation function to distinguish *good* solutions from the *bad* ones. For candidate solutions h_1 and h_2, we would like to know $time(t,h_1)$ and $time(t,h_2)$ to evaluate the candidates. Obviously, this is not possible, since getting these values would require to actually solve the problem.

For using meta-heuristics to solve the planning problems directly, there are following issues that need to be resolved:

- MCTS Simulations

  When MCTS selects the most urgent leaf, it starts a *simulation* to evaluate that leaf. Such simulation should lead to some goal state, where the resulting plan could be evaluated. However, reaching a goal state from some given initial state is equivalent to satisficing planning, which is a difficult problem in general.

  We don't require the simulation to be an optimal plan - suboptimal solutions are completely sufficient in the simulation phase - but we need the simulations to be very fast. In other words, we need means to finding suboptimal plans very quickly.

- Genetic Algorithms' crossover Operator

  If we used GA for planning, it would operate directly on the set of plans. During the search, GAs use crossover operator which takes two candidate solutions and combines them to produce another one.

  It is, however, difficult to guarantee that two valid plans will produce a valid plan during the crossover.

To sum up, the main research problems we are facing are:

1. Stopping criterion of the preprocessing
2. Evaluation function for the preprocessing
3. MCTS simulations
4. Search operators that combine valid solutions to different but still valid solutions
5. Overall design of a hyper-heuristic based planner

7 STATE OF THE ART

We provide an overview of the state of the art to all previously mentioned research topics.

7.1 Stopping Criterion of the Preprocessing

Published papers on preprocessing of planning problems like (Edelkamp, 2006; Haslum et al., 2007) use very simple stopping criteria - number of steps without improvement or a fixed number of steps. These techniques don't concern themselves with any reasoning about proper distribution of computation time either.

Matter of designing stopping criteria (or rather restarting criteria) is studied in the field of evolutionary optimization (Solano and Jonyer, 2007). Statistical methods already exist which we believe can be modified to be used in the preprocessing phase of planning problems.

7.2 Evaluation Function for the Preprocessing

Published papers on creation of pattern databases (Edelkamp, 2006; Haslum et al., 2007) use various approximations of $time(t,h)$ as a fitness function. In general, there is a theory of *Estimating search effort* which we can use to approximate the $time(t,h)$ value.

Estimating Search Effort (Korf et al., 2001) tries to predict how many nodes will A* or IDA* expand before finding a solution, how many nodes will it expand in the *i-th layer*, what the *average branching factor* is going to be and so on.

7.3 MCTS Simulations

We described the problem with MCTS simulations in detail in (Trunda, 2013; Trunda and Barták, 2013). Simulations work as random samples of the search space, they should be fast and simple. In typical applications, they are realized by performing random steps. In planning, however, performing random actions is not guaranteed to find a goal state and it's not even guaranteed to end.

In this phase, it is possible to make use of many standard planning techniques, for example heuristic estimators. Popular heuristics used in modern planners cover: Landmark-cut (Pommerening and Helmert, 2013), Linear programming-based heuristics (Pommerening et al., 2014), Pattern databases (Pommerening et al., 2013), Delete relaxation (Hoffmann, 2011) and others (Helmert and Domshlak, 2009).

Several attempts have already been made to use a random walk-based sampling for planning. The *Arvand* planner (Nakhost and Müller, 2009) proves this idea to be viable as it performs well on the IPC. Arvand carries out several random walks (with a fixed length) at the start of the search to find good initial solutions. If this attempt fails, it switches back to standard A*.

The problem of very fast suboptimal planning was recently addressed by ICP. The latest IPC introduced an *Agile track*, where the solution quality was not considered at all and the only criterion was the computation time required to find a plan. Most participating planners, however, used standard search techniques only in different configurations and a very few completely new approaches was introduced.

7.4 Search Operators that Combine Valid Solutions to Different but Still Valid Solutions

This problem has been intensively studied in the field of evolutionary optimization (Simon, 2013) and also several attempts have been made to use GAs directly for planning (Westerberg and Levine, 2000; Brie and Morignot, 2005). Most popular approach to solving this problem is by post processing - after creating the new candidate solution, it is checked for validity and if not valid, it is replaced by the nearest valid solution.

Another way of dealing with this problem is to introduce a transformation on the set of all candidate solution which would map the subset of valid solutions "together" and then the search would only operate on that subset. We believe that such transformations (sometimes called *indirect representations*) (Sebald and Chellapilla, 1998; Rothlauf, 2006) have a great potential to be used in optimization planning.

7.5 Overall Design of a Hyper-heuristic based Planner

So far, no competitive planning system based on hyper-heuristics has emerged. There are, however, portfolio-based planners, that use several different algorithms and a policy to choose form them. These policies are usually quite simple - several algorithms are run together until one of them finishes.

8 METHODOLOGY

We will here describe the methodology for solving the research problems mentioned at the end of section 6. We will use the following notation:

- t be a planning problem instance
- S be the set of all sequences of actions of t
- $P \subseteq S$ be the set of all *plans*
- $f : P \mapsto \mathbb{R}$ be the objective function to be minimized
- $solve(t,h)$ be a procedure to solve t with a helpful information h (as defined earlier) returning $p \in P$
- $time(t,h)$ be the time requirements of $solve(t,h)$
- H be the set of all possible values for h

Standard forward search planning techniques (like A*) operate on the set S. They start form short sequences trying to prolong them in order to achieve some $p \in P$. During the search, they use a heuristic distance estimator to guide the search.

Meta-heuristic optimization techniques (like GAs) operate on the set P (set of all possible solutions) and use $f(p)$ as a fitness function to evaluate $p \in P$. They assume that candidate solutions from P can be easily obtained.

Hyper heuristics, on the other hand, operate on the set H searching for solutions h. To evaluate the solution $h \in H$ they use $f(solve(t,h))$ as a fitness function. As a "side effect", they search for the solution to the original problem. Such approach has a distinct advantages against classical techniques as it is able to adapt the search strategy specifically to the problem instance.

We believe that standard forward search planning techniques are most suitable for domains where goal states are very sparse (i.e. $|P|$ is small) and finding some $p \in P$ among S is difficult or in cases where we have to guarantee optimality.

Meta-heuristics, on the contrary, should be effective on domains where goal states are dense (i.e. large $|P|$) and finding optimal solution would take too much time.

First, we would like to develop a meta-heuristic optimization algorithm and then use it as one of the components for a hyper-heuristic based planner.

8.1 Meta-heuristic Planning Algorithm

We would like to use standard Evolutionary Algorithm for optimization planning. The issue remains how to guarantee that search operators (like crossover and mutation) will produce valid plans.

We decided to solve this problem by a penalty function. We extend the function f to the whole S so that all $s \in S$ will be considered a valid solutions. We will devise means to evaluate invalid solutions in a way which would guide the search towards valid solutions (i.e. invalid solutions that are close to valid ones have a better evaluation that those that are far from any valid solution).

For this task, we will make use of heuristic distance estimators to tell us how far from some valid solution the candidate solution is. The new objective function $f' : S \mapsto \mathbb{R}$ will be a combination of f (which is a sum of costs of used actions) and a heuristic distance estimator d. d is zero for all $p \in P$ and greater than zero for $s \in S \setminus P$ (i.e. it penalizes invalid solutions).

The weights of f and d in the formula as well as the type of heuristic estimator used will be parameters of the algorithm. These parameters may later be subjected hyper-optimization.

8.2 Designing a Hyper-heuristic based Planner

We would like to design the hyper-heuristic planner using MCTS algorithm. The system should be able to find the most suitable search algorithm as well as manage the distribution of CPU time between the *search for searching strategy* and the *search for the solution*.

The system will be based on a portfolio of *low-level planning algorithms* which will be used in the simulation phase of MCTS. These low-level algorithms should have the following properties:

- be able to solve satisficing planning task - find a path to some goal state from given initial state
- be very fast
- may find (even vastly) suboptimal solutions
- the portfolio should be diverse - for every planning domain there should be an algorithm that works well on that domain

- if the low-level algorithm requires some initial parameters to be set, then we create more copies of this algorithm and add them to the portfolio with different parameter settings

As those low-level planning algorithms, we will use:

- Standard planning algorithms - A*, IDA*, weighted-A*, enforced hill-climbing and others
- combined with standard heuristic distance estimators (mentioned in section 7.3)
- "non-standard" search algorithms - beam-stack search, symbolic search and others (Edelkamp and Schrdl, 2012)
- meta-heuristic optimization algorithms including the one described in the previous section
- planners from the *Agile track* of IPC

The overall MCTS algorithm should combine the selection of promising candidate plans and promising search strategies for given problem. We have already developed some concrete ideas of how to do that and we will describe one of them here.

We will use MCTS in a standard way for planning as in (Trunda, 2013; Trunda and Barták, 2013), that is:

- tree covers an initial part of the problem state-space
- root of the tree represents an initial state
- edges from the node correspond to applicable actions
- successors correspond to states after applying the action
- each node represents a sequence of actions given by labels of edges on the path from root to the node

In the figure 3 there is an example of MCTS tree early in the search. s0 is the initial state, s1 is the selected leaf, a_1 to a_3 are actions. In the figure 4 there is the tree after expansion. New states that are reachable from s1 are added.

The algorithm works as described in section 2.3. It selects the most urgent leaf, expands it by adding its successors to the tree and then runs a simulation from this leaf. Simulations corresponds to finding some path from a state that the leaf represents to some goal state.

We will enhance this tree in a following manner: to every leaf node, we add new successors - one for each low-level planner in the portfolio. We will call them *virtual leaves*. The selection phase will work in the same way and select the most urgent virtual leaf - which means that is selects the (real) leaf and then

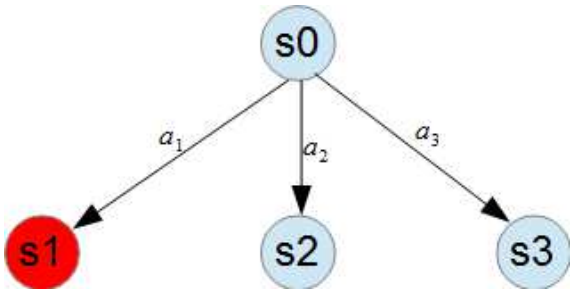

Figure 3: Example of a classic MCTS tree before expansion.

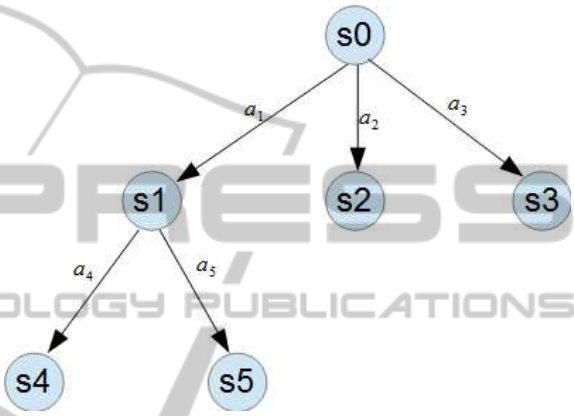

Figure 4: Example of a classic MCTS tree after expansion.

an algorithm to use. During the simulation phase, the selected algorithm will be used. After the expansion, however, the virtual leaves will not remain in the tree as inner nodes, but will move to the new leaves.

In the figure 5 there is an example of an enhanced MCTS tree. s0 is the initial state, s1 to s3 are other states. a_1 to a_3 are actions and *Alg1* to *Alg3* are virtual leaves, *Alg2* of s2 is the selected leaf. In the figure 6 there is the tree after expansion. New states that are reachable from s2 are added, but virtual leaves are not kept as inner nodes in the tree. The are copied to the successors together with all statistical information they were holding.

This way only the real nodes remain in the tree (therefore saving space), but the algorithm is still able to use different search algorithms in different parts of the tree. Inner nodes will accumulate all the simulation results no matter of the low-level algorithm that was used. This behaviour is desired, since the simulation should be random and the results of any of the low-level algorithms could theoretically by generated by a random walk so it makes sense to accumulate the results.

This design should allow the algorithm to automatically decompose the domain and use different search algorithms in different components. The node selection policy should prefer the most suitable algo-

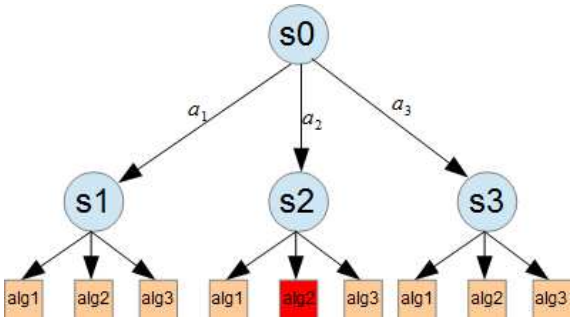

Figure 5: Example of an enhanced MCTS tree before expansion.

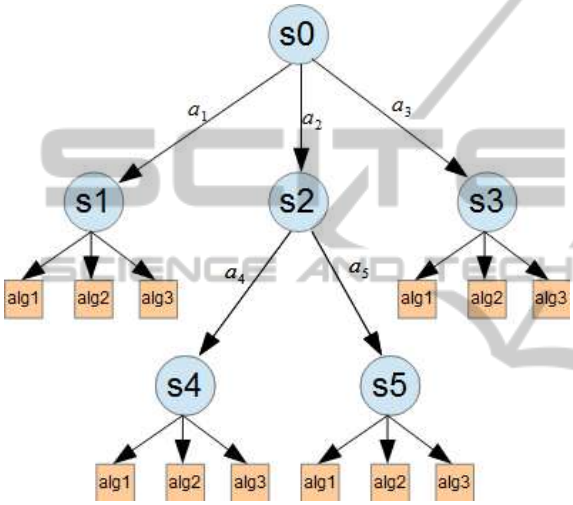

Figure 6: Example of an enhanced MCTS tree after expansion.

rithm in each part of the tree and therefore support division of the tree into parts representing different components of the domain.

This is just a basic idea which needs to be implemented and properly tested. It is likely that during our research we will make changes to the design.

8.2.1 CPU Time Management

It is important to keep track of how long the simulations take. In the properties that the low-level algorithms should have, we mentioned speed, but it is not possible to guarantee a priori that an algorithm will be fast on all domains. If we selected wrong algorithm, the single simulation might take more time than finding an optimal solution by the right algorithm.

We propose the following solution: we set a time limit on how long the simulation can take. This limit will be low on the beginning and will increase in time to allow more sophisticated algorithms in the simulation phase.

Furthermore, during the selection phase of the virtual leaves, we will consider not only solution quality (like with the inner nodes), but also time, that the simulations took. This should penalize the algorithms that found good solutions, but the search took too long.

Techniques where the search algorithm regulates itself during the search fall into category of *autonomous methods* (Hamadi et al., 2012) that are becoming popular these days. We believe that MCTS is a suitable platform for autonomous search and we would like to incorporate more of these techniques into the final design.

8.2.2 Evaluation of the Simulations

To evaluate the random sample from selected leaf l using the selected algorithm h, we need to get the value $f(solve(l,h))$. One way to do that is to actually run the search in the simulation phase. It may, however, happen on difficult domains that no method will be able to finish even one simulation within the time limit described in the previous subsection. And even if it does, the search might take quite a long time and be the bottle-neck of the algorithm.

Therefore it might be helpful to devise another means of random sampling. We would like to create a surrogate model based on fitness approximation techniques (Shi and Rasheed, 2010) which would allow us to evaluate the samples much faster. Also for algorithms like A*, the *Estimating search effort theory* (mentioned earlier) can be used.

Another interesting approach for determining which algorithm is the best for given problem is the *Sub-sampling principle* that we successfully used for example in (Trunda and Barták, 2014). It is based on an assumption, that algorithms that are good on small problems will also be good on larger problem from the same domain a vice versa - if an algorithm is bad on small problem, it will still be bad on a larger problem from the same domain. We can therefore test the algorithm on small problems (which is much faster) and transfer the results to original problem instances. We can easily create small problems as a subproblems of the original.

9 EXPECTED OUTCOME

The outcome of the PhD. thesis should be new theoretical and practical results about using the methods of soft-computing in planning. Specifically:

- creation of a new planning system based on optimization meta-heuristics

- introduction of the hyper-heuristic principle to planning, creation of a stronger planner than simple portfolios
- contribution to *algorithm selection* problem in planning (especially identifying meta-features of search problems)

ACKNOWLEDGEMENT

The research is supported by the Grant Agency of Charles University under contract no. 390214 and it is also supported by SVV project number 260 104.

REFERENCES

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Brie, A. H. and Morignot, P. (2005). Genetic planning using variable length chromosomes. In Biundo, S., Myers, K. L., and Rajan, K., editors, *ICAPS*, pages 320–329. AAAI.

Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. (2008). Monte-carlo tree search: A new framework for game ai. In *Proceedings of the 4th Artificial Intelligence for Interactive Digital Entertainment conference (AIIDE)*, pages 216–217. AAAI Press.

Edelkamp, S. (2006). Automated creation of pattern database search heuristics. In Edelkamp, S. and Lomuscio, A., editors, *MoChArt*, volume 4428 of *Lecture Notes in Computer Science*, pages 35–50. Springer.

Edelkamp, S. and Schrdl, S. (2012). *Heuristic Search - Theory and Applications*. Academic Press.

Genesereth, M. R., Love, N., and Pell, B. (2005). General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, Amsterdam.

Hamadi, Y., Monfroy, E., and Saubion, F. (2012). *Autonomous search*. Springer-Verlag.

Haslum, P., Botea, A., Helmert, M., Bonet, B., and Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, pages 1007–1012. AAAI Press.

Helmert, M. and Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A., Howe, A. E., Cesta, A., and Refanidis, I., editors, *ICAPS*. AAAI.

Hoffmann, J. (2011). Where Ignoring Delete Lists Works, Part II: Causal Graphs. In *21st International Conference on Automated Planning and Scheduling*, Freiburg, Allemagne.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the 15th European Conference on Machine Learning (ECML)*, pages 283–293. Springer Verlag.

Korf, R. E., Reid, M., and Edelkamp, S. (2001). Time complexity of iterative-deepening-a*. *Artif. Intell.*, 129(1-2):199–218.

Nakhost, H. and Müller, M. (2009). Monte-carlo exploration for deterministic planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1766–1771.

Olaya, A., López, C., and Jiménez, S. (visited December 10, 2014). International planning competition. [online].

Pommerening, F. and Helmert, M. (2013). Incremental lm-cut. In Borrajo, D., Kambhampati, S., Oddi, A., and Fratini, S., editors, *ICAPS*. AAAI.

Pommerening, F., Rger, G., and Helmert, M. (2013). Getting the most out of pattern databases for classical planning. In Rossi, F., editor, *IJCAI*. IJCAI/AAAI.

Pommerening, F., Rger, G., Helmert, M., and Bonet, B. (2014). Lp-based heuristics for cost-optimal planning. In *ICAPS*. AAAI.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms (2. ed.)*. Springer.

Rothlauf, F. (2011). *Design of Modern Heuristics*. Natural Computing Series. Springer.

Schadd, M. P. D., Winands, M. H. M., van den Herik, H. J., Chaslot, G. M. J.-B., and Uiterwijk, J. W. H. M. (2008). Single-player monte-carlo tree search. In *Proceedings of the 6th international conference on Computers and Games (CG '08)*, volume 5131 of *LNCS*, pages 1–12. Springer Verlag.

Sebald, A. V. and Chellapilla, K. (1998). On making problems evolutionarily friendly - part 2: Evolving the most convenient coordinate systems within which to pose (and solve) the given problem. In Porto, V. W., Saravanan, N., Waagen, D. E., and Eiben, A. E., editors, *Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 281–290. Springer.

Shi, L. and Rasheed, K. (2010). A survey of fitness approximation methods applied in evolutionary algorithms. In Tenne, Y. and Goh, C.-K., editors, *Computational Intelligence in Expensive Optimization Problems*, volume 2 of *Adaptation Learning and Optimization*, pages 3–28. Springer Berlin Heidelberg.

Simon, D. (2013). *Evolutionary Optimization Algorithms*. Wiley.

Solano, M. and Jonyer, I. (2007). Performance analysis of evolutionary search with a dynamic restart policy. In Wilson, D. and Sutcliffe, G., editors, *FLAIRS Conference*, pages 186–187. AAAI Press.

Toropila, D., Dvořák, F., Trunda, O., Hanes, M., and Barták, R. (2012). Three approaches to solve the petrobras challenge: Exploiting planning techniques for solving real-life logistics problems. In *Proceedings of 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 191–198. IEEE Conference Publishing Services.

Trunda, O. (2013). Monte carlo techniques in planning. Master's thesis, Faculty of Mathematics and Physics, Charles University in Prague.

Trunda, O. (2014). Automatic creation of pattern databases in planning. In Kurková, V., editor, *Proceedings of 14th conference ITAT 2014 Workshops and Posters*, volume 2, pages 85–92. Institute of Computer Science, AS CR.

Trunda, O. and Barták, R. (2013). Using monte carlo tree search to solve planning problems in transportation domains. In Castro, F., Gelbukh, A. F., and Gonzlez, M., editors, *MICAI (2)*, volume 8266 of *Lecture Notes in Computer Science*, pages 435–449. Springer.

Trunda, O. and Barták, R. (2014). Determining a proper initial configuration of red-black planning by machine learning. In *Proceedings of the International Workshop on Meta-learning and Algorithm Selection*, volume 1201, pages 51–52. CEUR Workshop Proceedings.

Vaquero, T. S., Costa, G., Tonidandel, F., Igreja, H., Silva, J. R., and Beck, C. (2012). Planning and scheduling ship operations on petroleum ports and platform. In *Proceedings of the ICAPS Scheduling and Planning Applications Workshop (SPARK)*, pages 8–16.

Westerberg, C. H. and Levine, J. (2000). "genplan": Combining genetic programming and planning. In Garagnani, M., editor, *19th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2000)*, The Open University, Milton Keynes, UK.