Using Monte Carlo Tree Search to Solve Planning Problems in Transportation Domains

Otakar Trunda and Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 25, Praha, Czech Republic
otaTrunda@gmail.com, bartak@ktiml.mff.cuni.cz

Abstract. Monte Carlo Tree Search (MCTS) techniques brought fresh breeze to the area of computer games where they significantly improved solving algorithms for games such as Go. MCTS also worked well when solving a real-life planning problem of the Petrobras company brought by the Fourth International Competition on Knowledge Engineering Techniques for Planning and Scheduling. In this paper we generalize the ideas of using MCTS techniques in planning, in particular for transportation problems. We highlight the difficulties of applying MCTS in planning, we show possible approaches to overcome these difficulties, and we propose a particular method for solving transportation problems.

Keywords: planning, search, Monte Carlo, logistic, transportation.

1 Introduction

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 modeling 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. Currently, the most efficient approach to solve planning problems is heuristic forward search. The paper [13] showed that classical planners are not competitive when solving a real-life transportation planning problem of the Petrobras company [15]. 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. This brought us to the idea of generalizing the MCTS algorithm to a wider class of planning problems. A sampling based approach has already been investigated in the field of planning [17] (in a simplified form). The Arvand planner [9] proved that the idea of using random-walks to evaluate states in deterministic planning is viable.

Monte Carlo Tree Search algorithm is a stochastic method originally proposed for computer games. MCTS was modified for a single-player games and it is also applicable to optimization problems. However, there are still difficulties when applying to planning problems, namely existence of infinite sequences of actions and dead-ends. In this paper we identify these difficulties and we discuss possible ways to overcome them. We then show how to modify the MCTS algorithm to be applicable to a planning problem specification enhanced by so called meta-actions. We demonstrate this idea using transportation problems which are natural generalizations of the Petrobras domain [15].

The paper is organized as follows. We will first give a short background on planning and Monte Carlo Tree Search techniques and we will highlight possible problems when applying MCTS in planning including a discussion how to resolve these problems. We will then characterize the transportation planning domains and show how to identify them automatically. After that we will describe the modifications necessary to apply MCTS to solve planning problems in transportation (and possible other) domains. The paper will be concluded by experimental comparison of our approach with the LPG planner [6].

2 Background

2.1 Planning

In this paper we deal with classical planning problems, that is, with finding a sequence of actions transferring the world from a given initial state to a state satisfying certain goal condition [5]. *World states* are represented as sets of predicates that are true in the state (all other predicates are false in the state). For example the predicate $at(r1, l1)$ represents information that some object $r1$ is at location $l1$. *Actions* describe how the world state can be changed. Each action a is defined by a set of predicates $prec(a)$ as its precondition and two disjoint sets of predicates $eff^+(a)$ and $eff^-(a)$ as its positive and negative effects. Action a is applicable to state s if $prec(a) \subseteq s$ holds. If action a is applicable to state s then a new state $\gamma(a, s)$ defines the state after application of a as

$$\gamma(a, s) = (s \cup eff^+(a)) - eff^-(a)$$

Otherwise, the state $\gamma(a, s)$ is undefined. The goal g is usually defined as a set of predicates that must be true in the goal state. Hence the state s is a *goal state* if and only if $g \subseteq s$.

The *satisficing planning task* is formulated as follows: given a description of the initial state s_0, a set A of available actions, and a goal condition g, is there a sequence of actions $(a_1, \ldots, a_n)$, called a *solution plan*, such that $a_i \in A$, a_1 is applicable to state s_0, each a_i s.t. $i > 1$ is applicable to state $\gamma(a_{i-1}, \ldots \gamma(a_1, s_0))$, and $g \subseteq \gamma(a_n, \gamma(a_{n-1}, \ldots \gamma(a_1, s_0)))$?

Assume that each action a has some cost $c(a)$. An *optimal planning task* is about finding a solution plan such that the sum of costs of actions in the plan is minimized. Formally, the task is to find a sequence of actions $(a_1, \ldots, a_n)$, called an *optimal plan*, minimizing $\sum_{i=1}^{n} c(a_i)$ under the condition $g \subseteq \gamma(a_n, \gamma(a_{n-1}, \ldots \gamma(a_1, s_0)))$. In this paper we deal only with the optimal planning task so for brevity we will be talking about planning while we will mean optimal planning.

In practice, the planning problem is typically specified in two components: a planning domain and a planning problem itself. The *planning domain* specifies the names of predicates used to describe world states. For example, $at(?movable, ?location)$ means that we can use relations at between movable objects and locations (typing is used to classify objects/constants). Similarly, instead of actions the planning domain describes so called *operators* that are templates for actions. A particular action is obtained by substituting particular objects (constants) to the operator. We will give examples of operators specified in the PDDL language later in the text. The *planning problem* then specifies a particular goal condition and an initial state and hence it also gives the names of used objects (constants).

2.2 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. [11] 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 [3].

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* [1,7] which provide a theoretical background to measure quality of policies. We will present here the tree policy for the single-player variant of MCTS (SP-MCTS) due to Schadd et al. [11] that is appropriate for planning problems (planning can be seen as a single-player game where moves correspond to action selection).

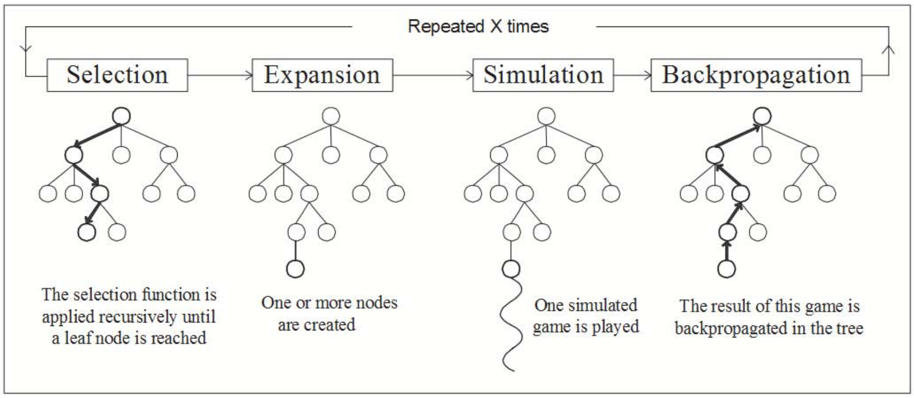

Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Fig. 1. Basic schema of MCTS [3]

Let $t(N)$ be the number of simulations/samples passing the node N, $v_i(N)$ be the value of i-th simulation passing the node N, and $\bar{v}(N)$ be the average value of all simulations passing the node N:

$$\bar{v}(N) = \frac{\sum_{i=1}^{t(N)} v_i(N)}{t(N)}$$

The SP-MCTS tree policy suggests to select the children node N_j of node N maximizing the following function:

$$\bar{v}(N_j) + C \cdot \sqrt{\frac{2\ln(t(N))}{t(N_j)}} + \sqrt{\frac{\sum_{i=1}^{t(N_j)} v_i(N_j)^2 - t(N_j) \cdot \bar{v}(N_j)^2}{t(N_j)}}$$

The first component of the above formula is a so called *Expectation* and it describes an expected value of the path going through a given node. This supports exploitation of accumulated knowledge about quality of paths. The second component is a *Bias*. The Bias component of a node N_j slowly increases every time the sibling of N_j is selected (that is every time the node enters the competition for being selected but it is defeated by another node) and rapidly decreases every time the node N_j is selected, that is the policy prefers nodes that have not been selected for a long time. This supports exploration of unknown parts of the search tree. *Bias* is weighted by a constant C that determines the *exploration vs. exploitation* ratio. Its value depends on the domain and on other modifications to the algorithm. For example in computer Go the usual value is about 0.2. When solving optimization problems, the range of values for the *Expectation* component is unknown opposite to computer games, where *Expectation* is between 0 (loss) and 1 (win). Nevertheless it is possible to use an adaptive technique for adjusting the parameter C in order to keep the components in the formula (*Expectation* and *Bias*) of the same magnitude [2]. The last component of the evaluation formula is a standard deviation and it was added by Schadd et al. [11] to improve efficiency for single-player games (puzzles).

3 MCTS for Planning

The planning task can be seen as the problem of finding a shortest path in an implicitly given state space, where transitions/moves between the states are defined by the actions. The reason why classical path-finding techniques cannot be applied there is that the state space is enormous. From this point of view planning is very close to single-player games though there are some notable differences.

3.1 Cycles in the State-Space

MCTS uses simulations to evaluate the states. Hence, from the planning perspective, we need to generate solution plans – valid sequences of actions leading to a goal state. Unlike the SameGame and other game applications of MCTS, planning problems allow infinite paths in the state-space (even though the state-space is finite) and this is quite usual in practice since the planning actions are typically reversible. This is a serious problem for the MCTS algorithm since it causes the expected length of the simulations to be very large and therefore only a few simulations can be carried out within a given time limit. There are several ways how to solve this problem.

1. *Modifying the state-space* so that it does not contain infinite paths. In general this is a hard problem itself as it involves solving the underlying satisficing planning problem, which is intractable.
2. *Using a simulation heuristic* which would guarantee that the simulation will be finite and short. Such a heuristic is always a contribution to the MCTS algorithm since it makes the simulations more precise. Obtaining this heuristic may require a domain-dependent knowledge, but there exist generally applicable planning heuristics.
3. *Setting an upper bound* on the length of simulations. This approach has two disadvantages: the upper bound has to be set large enough so that it would not cut off the proper simulations. Still if most of the simulations end on the cut-off limit, then every one of them would take a long time to carry out which would have a bad impact on the performance. Furthermore it is not clear how to evaluate the cut-off simulations.

3.2 Dead-Ends

The other problem is existence of plans that do not lead to a goal state. We use the term *dead-end* to denote a state such that no action is applicable to this state and the state is not a goal state. Note that dead-ends do not occur in any game domain since in games any state that does not have successors is considered a goal state and has a corresponding evaluation assigned to it (like Win, Loss, or Draw in case of Chess or Hex, or a numerical value in case of SameGame for example). In planning, however, the evaluation function is usually defined only for the solution plans leading to goal states. A plan that cannot be extended doesn't necessarily have to be a solution plan and it is not clear how to evaluate the simulation that ended in a dead-end. Possible ways to solve this issue are:

1. *Modifying the state-space* so that it would not contain dead-ends.
2. *Using a simulation heuristic* which would guarantee that the simulation never encounters a dead-end state.
3. *Ignoring the simulations* that ended in dead-end states. If the simulation should end in a dead-end we just forget it and run another simulation (hoping that it would end in a goal state and gets properly evaluated). This approach might be effective if dead-ends are sparse in the search space. Otherwise we could be waiting very long until some successful simulation occurs.
4. Finding a way to *evaluate the dead-end states.*

3.3 Dead Components

A *dead component* is a combination of both previous problems – it is a strongly connected component in the state-space that does not contain any goal state. This is similar to the dead-ends problem except that we can easily detect a dead-end (since there are no applicable actions there) but it is much more difficult to detect a dead component since we would have to store all visited states (and search among them every time) which would make the simulation process much slower and the algorithm less effective.

4 Transportation Domains

As we mentioned earlier, modifying the planning domain so that its state-space wouldn't contain infinite paths and dead-end states requires to actually *solve* the underlying satisficing problem. The MCTS algorithm has been used to solve satisficing problems such as Sudoku however the method is much better suited for optimization problems. Since the problem of satisficing planning is difficult and in general intractable we have decided to restrict the class of domains which our planner addresses. Based on good results with the Petrobras domain [13], we chose to work with the transportation domains. This kind of domains seems to be well suited for the MCTS method since:

1. it is naturally of an optimization type (typically fuel and time consumption are to be minimized),
2. the underlying satisficing problem is usually not difficult,
3. transportation problems frequently occur in practice.

Knowing that the domain is of the transportation type we can use *domain analysis techniques* to gather more information about the domain structure and dynamics. This higher level information is then exploited during the planning process/simulation (as described in the next chapter).

We introduce the term *Transportation component* denoting a part of the planning domain that has a typical *transportation structure*. The transportation component describes some *vehicles*, *locations*, and *cargo* and specifies actions for *moving* the vehicles between the locations and for transporting the cargo by *loading* and *unloading* it by the vehicles. We also assume that the goal is to deliver cargo to specific destinations. We have created a template that describes such a structure – the template describes *relations* between the *symbols* that are typical for the transportation domains. By *symbols* we mean names of the predicates, types of the constants, and names of the operators.

For example consider the following two operators (specified in PDDL) originating from two different planning domains:

```
(:action load
  :parameters    (?a - airplane ?p - person
                  ?l - location)
  :precondition (and (at ?a ?l) (at ?p ?l))
  :effect        (and (not (at ?p ?l))
        (in ?p ?a)))

(:action get
  :parameters    (?v - vehicle ?c - cargo
                  ?d - destination)
  :precondition (and (isCargoAt ?c ?d)
        (isVehicleAt ?v ?d))
  :effect        (and (not (isCargoAt ?c ?d))
        (isInside ?c ?v))))
```

Even though these operators use different predicates, relations between the symbols *(get, vehicle, cargo, destination, isCargoAt, isVehicleAt, isInside)* are exactly the same as the relations between *(load, airplane, person, location, at, at, in)*.

The example shows that the *structure* of the domain (which is what we want to capture) does not depend on the symbols used but only on the relations between the symbols. This gives us means to define a *generic transportation structure* and then we can check whether some given domain *matches* this predefined structure. For example, the action *get* in the above example can be seen as a *template of loading* and we can say that the action *load* matches this template.

A *transportation component* is defined by the names of operators for loading (denoted *Op-L*), unloading (*Op-U*) and moving (*Op-M*) actions, the names of predicates describing positions of the vehicles (denoted *Veh-At*) and of the cargo (*Carg-At*, *Carg-In*), as well as the types of constants that represent vehicles (*Type-Veh*), cargo (*Type-Carg*) and locations (*Type-Loc*). Moreover there has to be a certain relationship between these symbols. Operator *Op-L* has to have a predicate with the name *Cargo-At* among its positive preconditions and negative effects and a predicate with the name *Cargo-In* among its positive effects. Both these predicates has to share some variable that has to have a type *Type-Cargo*. Also it has to have a predicate with the name *Veh-At* in its positive preconditions and this predicate has to share some variable with the predicate *Cargo-In* mentioned above. This variable has to have a type *Type-Veh*. Finally the two predicates with names *Veh-At* and *Cargo-At* in the operator definition have to share some variable that has a type *Type-Loc*. In the above example of two loading operators, we have the following matchings:

Op-L	load	get
Veh-At	at	isVehicleAt
Cargo-At	at	isCargoAt
Cargo-In	in	isInside
Type-Veh	airplane	vehicle
Type-Carg	person	cargo
Type-Loc	location	destination

The reader can easily verify that the required relations between the symbols hold. In a similar way we can define the templates for *unloading* and *moving* operators. A complete description of all the variables and constraints of the templates as well as more examples of transportation components can be found in [14].

We say that a planning domain contains a transportation component if it is possible to substitute symbols from the domain description (names of the operators, predicates, and types) to all three operator templates such that all the constraints hold. It is possible that the domain contains more types of vehicles or cargo or more operators for loading, unloading, and moving. For example in the Zeno-Travel domain [10] the planes can travel at two different speeds – the domain contains two different operators for moving - *Fly* and *Fly-Fast*. In these cases we say that the domain contains more than one transportation component. It is also possible that the transportation component is embedded within a more complex structure in the planning domain. For example we may assume that *hoist* is necessary to load cargo to a vehicle. Then the predicates may have arity larger than two (*hoist* holds *cargo* at given *location*), the operators may have more than three parameters (hoist is included), and they may have more preconditions (vehicle is empty) and effects (hoist will be empty after the action, while the vehicle will be full). Such operators can still fit the template as we do not require the domain to be *isomorphic* to the template but rather only to *match* the template.

Notice that the transportation component is defined by the values of some variables which have to satisfy certain constraints defined by the template. Hence, we can search for the components automatically in the domain description using constraint programming (CP) techniques [4]. The names of operators (like *Op-L* or *Op-M*), the names of the predicates (like *Veh-At* or *Carg-In*) and the types (like *Type-Loc*) in the definition of the transportation component can be seen as CP variables. The domains of these variables are derived from the planning domain description. For example the domain of the variable *Op-L* is a set of all names of operators defined in the planning domain, the domain of the variable *Type-Veh* is a set of all types used in the planning domain and so on. Finding all the transportation components is equivalent to finding all the solutions of this Constraint Satisfaction Problem. This step might be computationally demanding but it is done only once as preprocessing before the actual planning starts. The transportation component is defined by relations between symbols including types. Therefore this domain analysis technique can only be used on domains that support typing. This however is not a restriction since the types can be automatically derived from the domain description even if they are not given explicitly [16].

5 MCTS in Planning for Transportation Domains

Recall that one of the main components of the MCTS algorithm is simulation – looking for a solution plan from a given state. As these simulations are run frequently to evaluate the states, it is critical to make the simulations short (i.e. with low expected number of steps/actions) and to ensure that most of the simulations will reach the goal (i.e. low probability of reaching a dead-end or a dead component) so that the sampling would be fast and effective. We achieve this by modifying the state space based on the analysis of the planning domain.

We modify the planning domain by replacing the given action model with so called meta-actions (will be explained later). Meta-actions are learned during planning and their purpose is to rid the domain of cycles and dead-ends. The proposed model of meta-actions is theoretically applicable to all types of domains however in our planner we only use it for a specific type of planning domains where the learning technique is reasonably fast.

5.1 Meta-actions

Meta-actions (or composite actions) consist of sequences of original actions. The purpose of using Meta-actions instead of the original actions is to prevent some paths in the state-space from being visited during the simulation phase of the MCTS algorithm. The original action model allows *every* possible path in the search space to be explored by the planner though many paths are formed of meaningless combinations of actions which do not lead to the goal. For example the sequence $load - unload - load - unload - \ldots$ is a valid plan, but does not bring us closer to the goal. If the simulation visits such a path, it might spend a lot of time in cycles or end up in a dead-end. We use the meta-actions to eliminate as many meaningless paths as possible which makes the Monte-Carlo sampling process efficient enough to be worth using.

Each meta-action represents a sequence of original actions. If we want to apply the meta-action, we apply all the actions in the sequence successively. If we represent the state-space as a directed graph where vertices represent states and edges represent actions then the meta-actions correspond to paths in this graph. Our modification of the state-space can be seen as follows:

1. Identify important paths in the graph (learn the meta-actions).
2. Remove all original edges from the graph (we no longer use the original actions once we learned the meta-actions).
3. Add new edges to the graph. Edge is added from u to v if there exists a meta-action A such that $v = \gamma(A, u)$.

The planning/simulation is then performed on this new search space. Notice that by removing the original actions, we may effectively remove some valid plans from being assumed. Our goal is to create the meta-actions in such a way that the important paths in the state-space would be preserved (especially paths representing optimal solutions should be preserved) and paths leading to cycles or dead-ends would be eliminated.

5.2 Example of the Meta-actions Model

We will first give examples of two meta-actions constructed for the Zeno-Travel Domain [10]. The original domain description contains actions *Fly*, *Load* and *Unload*. The following two meta-actions were learned by the planner (the learning method will be described later).

```
(:action fly+load
 :parameters    (?a - airplane ?p - person
         ?from ?to - location)
 :precondition  (and (at ?a ?from)
               (at ?p ?to))
 :effect (and (not (at ?a ?from))
     (not (at ?p ?to)) (at ?a ?to)
     (in ?p ?a)))

(:action fly+unload
 :parameters (?a - airplane ?p - person
          ?from ?to - location)
 :precondition  (and (at ?a ?from)
               (in ?p ?a))
 :effect (and (not (at ?a ?from))
     (at ?a ?to) (not (in ?p ?a))
     (at ?p ?to)))
```

In this example the original state-space contains many infinite paths – for example performing action *fly* without ever loading anything or repeatedly performing actions *load-unload*. The modified state-space which uses meta-actions doesn't contain these paths. It still contains some infinite paths but their number is greatly reduced.

5.3 Learning the Meta-actions

Let us suppose that we have already found the transportation components in the planning domain. Every meta-action we create belongs to some transportation component and we shall describe the learning algorithm using the terms from the definition of a transportation component. In particular, we will use terms such as *vehicle*, *cargo*, *loaded cargo*, *operator Move* and so on even though we do not know the exact symbols that play these roles in the particular planning domain (for example *Move* can be called *fly* in the particular domain). The operators, types, and predicates might have different names in each particular transportation planning domain but their meaning is still the same. The term *operator Load* for example refers to the operator that *plays the role of loading* in the particular domain. We can use this assumption since we have already assigned meaning to these symbols by identifying the transportation components.

To recognize the important paths we use a landmark-based approach where we first find the state landmarks in the domain (*landmark* is a set of states such that every solution plan has to visit at least one of these states) and then we find paths between the nearest landmarks. These paths will be stored in a form of meta-operators and used during the simulation phase of MCTS (instantiated to meta-actions).

Assuming the transportation domains where the goal is to deliver some cargo simplifies the process of identifying landmarks. Basically, we have two types of landmarks. In order to deliver cargo, it has to be loaded in some vehicle and then unloaded. For every cargo that has not yet been delivered the set of all states where this cargo is loaded in some vehicle represents a state landmark of the problem. Similarly for a cargo that is loaded the set of all states where this cargo is not loaded represents a state landmark. We create the meta-actions by finding the shortest paths between two consecutive landmarks. For finding the shortest paths we use an exhaustive search where we assume that the domain is simple enough for this process to be tractable - i.e. the paths between the landmarks are short and simple. Having the sequence of actions, we do lifting to obtain a sequence of operators that then define the meta-operator.

Let us summarize the whole learning technique. Suppose that during the simulation phase the MCTS algorithm visits a state where no known meta-action is applicable. Then the learning procedure is initiated which works as follow.

1. select some transportation component randomly
2. select some not delivered cargo from this component randomly
3. if this cargo is not loaded, find the shortest path (paths) to some state where it is loaded; otherwise (i.e. if it is already loaded) find the shortest path to some state where it is not loaded. These paths consist of original actions.
4. create meta-actions from these sequences, lift them to meta-operators, and store them for future use.

In the Zeno-Travel domain these shortest paths only contained two actions and the meta-actions were fly+load and fly+unload. In the Petrobras domain however the paths might be longer. In this case meta-actions of a length up to four were found, namely *undock+navigate+dock+load* and *undock+navigate+dock+unload.*

The method of exploiting meta-actions can be described as follows. We find optimal sub-plans for sub-goals that need to be achieved to reach the expected "big" goal (for example load yet-unloaded cargo by first flying to the cargo location and then loading it), we encapsulate these sub-plans into meta-actions and generalize the meta-actions to meta-operators by lifting, and finally we use these meta-operators when planning for the "big" goal. This can be in principle done for any planning domain, but identifying the landmarks and finding sub-plans is in general a hard problem.

6 Experimental Results

We have compared our technique with the LPG planner [6] – a state-of-the-art domain-independent optimal planner for PDDL2.2 domains and the top performer at IPC4 in plan quality. We used two domains for comparison: the *Zeno-Travel domain* as an example of an artificial IPC domain – one of the simplest transportation domains possible, and the *simplified Petrobras domain*, which is much closer to real practical problems. The simplified Petrobras domain is similar to the originally proposed Petrobras domain [15] with the difference that the cargo no longer has any weight assigned to it, loading and unloading takes a fixed amount of time (no longer depends on the weight) and the ships can hold arbitrarily many cargo (no longer have a capacity limit). Also the plan quality is only measured by the fuel consumption and makespan (the original domain used several other vaguely defined criteria).

We have generated 40 random problems from both domains with the increasing size of the problems. The LPG planner optimized the fuel consumption while the MCTS planner optimized weighted sum of both fuel consumption and makespan. Every problem had a time limit assigned according to its size. The smallest problems had the time limit of 5 minutes, the largest ones had a limit of 40 minutes. The best solutions found within the time limit are reported. The experiments were conducted on the same computer – Asus A8N-VM CSM with processor AMD Opteron (UP) 144 @ 1800 MHz (8 cores) and 1024MB of physical memory.

6.1 Zeno-Travel Domain

The results of the experiments with the Zeno-Travel domain are shown in Figure 2. Both planners were able to solve all the problems. In plan quality the LPG planner outperformed the MCTS planner by approximately 10 percent overall (measured by the sum of fuel consumption and makespan).

6.2 Simplified Petrobras

In the case of simplified Petrobras domain both planners were again able to solve all the problems within the given time limit. The results of experiments are shown in Figures 3(a), 3(b) and 4.

The LPG planner outperformed the MCTS in fuel consumption as Figure 3(a) shows. The MCTS planner on the contrary outperformed LPG in makespan

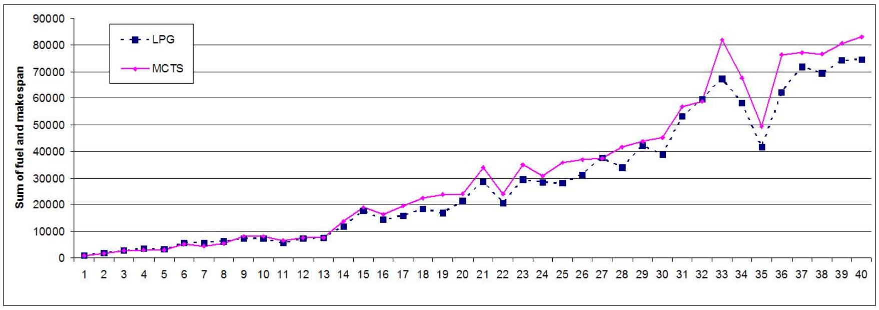
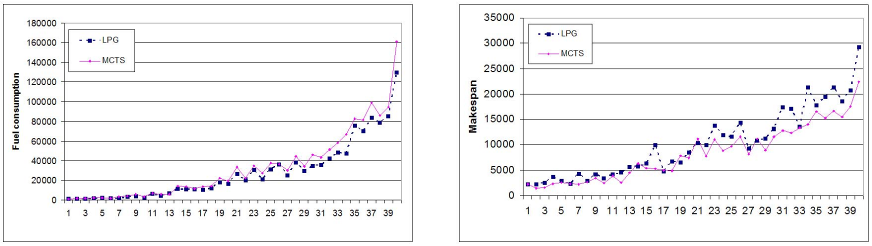


Fig. 2. Results of the experiments in the Zeno-Travel domain

(see Figure 3(b)). Figure 4 shows the results according to the weighted sum of both these criteria. We used the combined objective function $1 * fuel + 4 * makespan$ so that both criteria are comparable and equally important in the sum. In this case MCTS slightly outperformed the LPG by approximately 5 percent overall.



(a) Fuel consumption (b) Makespan

Fig. 3. Results of the experiments in the simplified Petrobras domain

6.3 Discussion

The preliminary experimental results show that the simple MCTS planner is competitive with the complex LPG planner. The LPG planner found simpler plans with less parallelism and less vehicles used. Therefore it had better fuel consumption but worse makespan. The MCTS planner on the contrary found more sophisticated plans using more vehicles and more actions.

The MCTS planner performed better on the more complicated domain (compared to LPG) which we believe is caused by the use of meta-actions. Meta-actions can capture complex paths in the state-space and allow to use these paths again during the planning. In the Zeno-Travel domain the planner only learned 2 meta-operators while in the simplified Petrobras the number of learned meta-operators was 12.

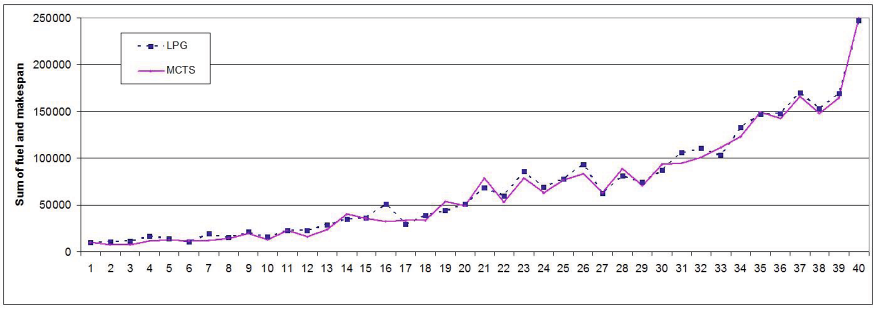


Fig. 4. Results of the experiments in the simplified Petrobras domain - weighted sum of makespan and fuel consumption

The MCTS technique combined with the learning of meta-actions proved to be competitive with standard planning software even though it is still only a prototype and still has a potential for further improvement. We believe that by efficient implementation, integration of other techniques [12] or by hybridization with other algorithms [8] the performance can be further increased.

7 Conclusions

In this paper we showed that an ad-hoc MCTS planner from [13] can be generalized to a wider range of planning domains while keeping its efficiency. We characterized transportation planning domains using templates of three typical operations (loading, moving, unloading), we showed how to automatically identify these operations in the description of any planning domain and how to exploit the structure found for learning macro-operations that speed-up MCTS simulations. The resulting MCTS planner is already competitive with the state-of-the-art planner LPG thought the implementation of the MCTS planner is not fine-tuned. Due to space limit, the paper focused on explaining the main concepts of the method, the formal technical details can be found in [14].

A universal MCTS planner would require a fast technique for solving the underlying satisficing problem which does not seem possible for arbitrary planning domain. It can however be used in the domains where the underlying problem is easy to solve and a complex objective function is used in the optimization problem. Transportation domains are a good example of such domains. We believe that there are many more planning domains that have this property (especially those that are practically motivated) and therefore MCTS techniques may become a new and efficient way to address such problems.

Acknowledgment. Research is supported by the Czech Science Foundation under the project no. P103-10-1287.

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning 47(2-3), 235–256 (2002)
2. Baudiš, P.: Balancing MCTS by Dynamically Adjusting Komi Value. ICGA Journal 34, 131–139 (2011)
3. Chaslot, G., Bakkes, S., Szita, I., Spronck, P.: Monte-Carlo Tree Search: A New Framework for Game AI. In: Proceedings of the 4th Artificial Intelligence for Interactive Digital Entertainment conference (AIIDE), pp. 216–217. AAAI Press (2008)
4. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers Inc. (2003)
5. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning: Theory and Practice. Elsiever Morgan Kaufmann, Amsterdam (2004)
6. Gerevini, A., Saetti, A., Serina, I.: Planning in PDDL2.2 Domains with LPG-TD. In: International Planning Competition, 14th International Conference on Automated Planning and Scheduling (IPC at ICAPS) (2004)
7. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
8. Loth, M., Sebag, M., Hamadi, Y., Schoenauer, M., Schulte, C.: Hybridizing Constraint Programming and Monte-Carlo Tree Search: Application to the Job Shop Problem (unpublished)
9. Nakhost, H., Müller, M.: Monte-Carlo exploration for deterministic planning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1766–1771 (2009)
10. Olaya, A., López, C., Jiménez, S.: International Planning Competition (2011), `http://ipc.icaps-conference.org/` (retrieved)
11. Schadd, M.P.D., Winands, M.H.M., van den Herik, H.J., Chaslot, G.M.J.B., Uiterwijk, J.W.H.M.: Single-player Monte-Carlo tree search. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 1–12. Springer, Heidelberg (2008)
12. Schadd, M.P.D., Winands, M.H.M., van den Herik, H.J., Aldewereld, H.: Addressing NP-Complete Puzzles with Monte-Carlo Methods. In: Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning vol. 9, (2008)
13. Toropila, D., Dvořák, F., Trunda, O., Hanes, M., Barták, R.: Three Approaches to Solve the Petrobras Challenge: Exploiting Planning Techniques for Solving Real-Life Logistics Problems. In: Proceedings of ICTAI 2012, pp. 191–198. IEEE Conference Publishing Services (2012)
14. Trunda, O.: Monte Carlo Techniques in Planning. Master's thesis. Faculty of Mathematics and Physics, Charles University in Prague (2013)
15. Vaquero, T.S., Costa, G., Tonidandel, F., Igreja, H., Silva, J.R., Beck, C.: Planning and scheduling ship operations on petroleum ports and platform. In: Proceedings of the ICAPS Scheduling and Planning Applications Workshop, pp. 8–16 (2012)
16. Wickler, G.: Using planning domain features to facilitate knowledge engineering. In: Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2011), pp. 39–46 (2011)
17. Xie, F., Nakhost, H., Müller, M.: A Local Monte Carlo Tree Search Approach in Deterministic Planning. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2011), pp. 1832–1833 (2011)