On Modeling Planning Problems: Experience from the Petrobras Challenge

Roman Barták[1] and Neng-Fa Zhou[2]

[1] Charles University in Prague,
Faculty of Mathematics and Physics
bartak@ktiml.mff.cuni.cz
[2] The City University of New York,
Brooklyn College
nzhou@acm.org

Abstract. The International Planning Competitions have led to development of a standard modeling framework for describing planning domains and problems – Planning Domain Description Language (PDDL). The majority of planning research is done around problems modeled in PDDL though there are only a few applications adopting PDDL. The planning model of independent actions connected only via causal relations is very flexible, but it also makes plans less predictable (plans look different than expected by the users) and it is probably also one of the reasons of bad practical efficiency of current planners ("visibly" wrong plans are blindly explored by the planners). In this paper we argue that grouping actions into flexible sub-plans is a way to overcome the efficiency problems. The idea is that instead of seeing actions as independent entities that are causally connected via preconditions and effects, we suggest using a form of finite state automaton (FSA) to describe the expected sequences of actions. Arcs in FSA are annotated by conditions guiding the planner to explore only "proper" paths in the automaton. The second idea is composing primitive actions into meta-actions, which decreases the size of a FSA and makes planning much faster. The main motivation is to give users more control over the action sequencing with two primary goals: obtaining more predictable plans and improving efficiency of planning. The presented ideas originate from solving the Petrobras logistic problem, where this technique outperformed classical planning models.

Keywords: planning, modeling, transportation, tabling.

1 Introduction

Recent research in the area of planning is centered on the representation of problems in the Planning Domain Description Language introduced for International Planning Competitions [2]. Having a standard modeling language accelerated research in the area of planning and led to development of many benchmark problems that are used to experimentally evaluate and compare various solving approaches. On the other hand, PDDL is based on the original STRIPS idea of having actions that are causally connected via their preconditions and effects. This makes planning very flexible but also introduces some undesirable behaviors. For example, an action for unloading an item can be planned immediately after an action that loaded the item. This is causally correct (unloading requires the item to be loaded which is achieved by the loading action) though from a human perspective such action sequences are not desirable (the state after unloading will be identical to the state before loading). It is possible to forbid such situations by changing the model in such a way that, for example, a transportation action is planned between loading and unloading[1]. However, such enhanced models are less natural and less readable by humans, and flaws can be easily introduced in the models if more such modifications are required. It seems more natural, if a human modeler prescribes possible (reasonable) action sequences. There exist two modeling approaches based on this idea, *hierarchical task networks* (HTNs) [9] and *timelines* [13]. While HTN uses the notion of task that decomposes into sub-tasks until primitive activities are obtained, timelines focus on modeling allowed time evolutions of state variables and synchronizations between the state variables.

In this paper we study a modeling framework positioned half way between timelines and HTNs. Similarly to [4] we propose to use a *finite state automaton* (FSA) describing allowed sequences of actions. A FSA plays the role of effects and conditions from classical planning as it says which actions may follow a given action. A FSA can be accompanied by additional constraints restricting when some transitions between the actions may occur. These conditions are different from classical action preconditions as they involve information about the goal (*Pickup* cargo only when there is some cargo to deliver). This is much closer to *control rules* [7], but rather than specifying control rules separately from the description of the planning domain, we suggest integrating them in the domain. This is an original idea of this paper.

We have found that the above modeling approach is not sufficient enough when solving real-life problems and we suggest additional extensions motivated by the Petrobras challenge. The Petrobras challenge is a logistic problem [15] of transporting cargo items between ports and oil platforms using vessels with limited capacity. The paper [14] studied three approaches to solve this problem. The winning technique was based on Monte Carlo Tree Search (MCTS) where the search was done over sequences of meta-actions. Each meta-action was then decomposed to primitive actions based on the situation. In this paper we use a very similar idea where we take "reasonable" sub-plans (sequences of actions) and encode them as a single meta-action. The FSA is then defined over these meta-actions. During planning the selected meta-action decomposes into a sequence of primitive actions depending on the current situation. There already exists a concept of *macro-actions* in planning [5]. However, while a macro-action decomposes to a fixed sequence of primitive actions, a meta-action may decompose to different sequences of primitive actions based on the situation. The concept of meta-actions is closer to HTNs, though we use only one level of decompositions – from a meta-action to a sequence of primitive actions. Also, planning in our framework is based on action sequencing rather than on task decomposition.

This paper shows that action grouping and prescribed action sequencing are very important for the Petrobras challenge. Instead of sophisticated MCTS method, we use backtracking accompanied by tabling [16] to solve the problem. As we shall experi-

[1] The action *unload* may use some new proposition – a semaphore – as its precondition, and this proposition is removed by the *load* action while added by the *transport* actions.

mentally show the resulting method achieves very similar performance to the MCTS algorithm. Hence we believe the presented modeling concepts are important for solving real-life planning problems and may bring significant efficiency boost.

The paper is organized as follows. We will first briefly introduce the Petrobras challenge, highlight some of its important components, and describe the three techniques already applied to this problem. Then we will present the proposed modeling framework based on finite state automata, explain the solving technique, and show how actions can be grouped to meta-actions. After that we will experimentally compare our method with the current best methods using the Petrobras benchmarks from [14]. Description of possible future directions of research will conclude the paper.

2 The Petrobras Challenge

2.1 Problem Formulation

International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012) brought several real-life motivated modeling and solving challenges including the Petrobras problem. The Petrobras problem [15] deals with planning deliveries of cargo items between ports and platforms while respecting the capacity limits of vessels, ports, and platforms. The ports and platforms can serve a limited number of vessels at the same time; the vessels can transport limited weight of cargo items, vessels' load influences vessels' speeds and fuel consumption, and the limited capacity of fuel tanks must also be assumed when planning transport (refueling is possible). Given a set of cargo items to deliver (including cargo weights and initial ports), the problem is to find a feasible plan that guarantees the delivery of all cargo items to given destinations (platforms) and respects the constraints on the vessel, port, and platform capacities. The vessels should leave a waiting area, perform their mission and go back to one of the waiting areas. Loading and unloading of cargo items are done at ports and platforms and require some time. The vessels can be refueled at ports and certain platforms and each vessel must always have enough fuel to go to the nearest refueling station. We will describe the particular primitive actions, which can be assumed during planning, later in the text.

2.2 Techniques Used to Solve the Challenge

So far there was only one attempt to solve the full Petrobras challenge. Toropila et al. [14] applied classical planning, temporal planning, and Monte Carle Tree Search (MCTS) to solve the Petrobras challenge.

The **classical planning approach** modeled the problem in PDDL 3.0 [10] with numerical fluents describing the restricted resource capacities (fuel tank, cargo capacity, ports/platforms docks). This model used actions as specified in the problem formulation, namely: *navigate-empty-vessel, navigate-nonempty-vessel, load-cargo, unload-cargo, refuel-vessel-platform, refuel-vessel-port, dock-vessel, undock-vessel.* SGPlan 6.0 [11] was used to solve the problem while optimizing fuel consumption (other planners have been tried but were not able to process the domain description). Action duration was added to the solution in the post-processing stage.

The **temporal planning approach** modeled the problem in PDDL 3.1 [12] with fluents and durative actions. Basically the same set of actions with durations was used. This model supports concurrency of actions directly. The Filuta planner [8] was used to solve the problem while optimizing makespan. Filuta uses ad-hoc solvers to handle resources, namely unary, consumable, and reservoir resources are supported.

The last approach exploited **Monte Carlo Tree Search** (MCTS) techniques that become recently popular in computer Go [6]. To allow using MCTS, a different action model was applied to obtain finite plans in search branches. This model is based on four abstract actions: *Load*, *Unload*, *Refuel*, *GoToWaitingArea*. These actions describe "intentions" and they are decomposed to particular actions based on situation (state). For example, the action *Unload* assumes that cargo is loaded and vessel is either in a port or a platform. This action is decomposed in the following way. If the current location of the vessel is the same as the target location of the cargo then only a single underlying action *unload-cargo* is performed, otherwise the abstract action is translated to the following sequence of the original actions: *undock-vessel*, *navigate-nonempty-vessel*, *dock-vessel*, *unload-cargo*.

The experiments with the challenge data and randomly generated data, where the number of vessels and cargo items varied (3-10 vessels, 1-15 cargo items), showed that the classical planning approach is not viable as it cannot solve problems with more (7+) cargo items. The MCTS was the clear winner, followed by Filuta that can solve all problems, but the quality of plans was significantly lower (30% for makespan, 130% for fuel consumption).

3 The Novel Modeling Approach

The approaches from the Petrobras challenge inspired us to explore the reasons of success of the MCTS technique (and partly Filuta). In particular we focused on the "predefined" action sequences that are hidden in the abstract actions of the MCTS approach and partly also in the special resource solvers in the Filuta planner. Another motivation for our research went from the efficient model of the Sokoban game implemented in B-Prolog [17]. This model was also based on grouping actions into specific sequences. Our hypothesis is that even a simple search algorithm, for example depth-first search with tabling [16], can solve complex planning problems provided that the model itself guides the solver by describing expected action sequences rather than giving only independent actions connected via causal relations.

3.1 Model Based on Finite State Automata

Let us first describe the possible plans of each vessel in the Petrobras challenge as a finite state automaton. Finite state automata (FSA) were shown to significantly improve efficiency of constraint-based planners [4] and they represent a natural model to describe allowed action sequences. FSAs are also used in the Filuta planner to describe allowed state transitions. Figure 1 shows the FSA that models all possible actions and transitions between the actions for a single vessel in the Petrobras domain.

We already use some abstract actions there, for example the action *navigate* means *navigate-empty-vessel*, if the vessel is empty, or *navigate-nonempty-vessel*, if some cargo is loaded to the vessel. Similarly the action *refuel* means either *refuel-vessel-platform* or *refuel-vessel-port* depending on whether the vessel is docked in a platform or in a port. Notice also that the presented FSA restricts some action sequences. In particular, it is not possible to dock immediately after undocking – there is no practical reason for such a sequence of actions though the classical PDDL model allows it. Similarly, refueling is done only after loading/unloading – this removes symmetrical sequences of actions with an identical overall effect (the vessel is loaded/unloaded and refueled). Finally, during one stop at the port or platform, the FSA allows either loading of cargo or unloading of cargo, but not both operations together. This is motivated by the particular problem to be solved – we need to move cargo from ports to platforms. Hence, there is no need to unload a cargo (at some platform) and load another cargo at the same location. In principle, it might be possible to move cargo to some intermediate location where it will be picked up by another vessel. However such flexible plans were not found necessary in the Petrobras challenge. Note finally that all these sequencing restrictions are naturally modeled using the transitions in the FSA. If more flexible plans are desirable then the corresponding transitions can be added to the FSA. We also use another mechanism to restrict sequencing by putting constraints on the transitions. These constraints describe situations when the transition is allowed. We will describe these constraints in more detail later in the text. The classical planning model with action preconditions and effects makes expressing such allowed action sequences much more complicated and not very natural (see the footnote 1 in the Introduction).

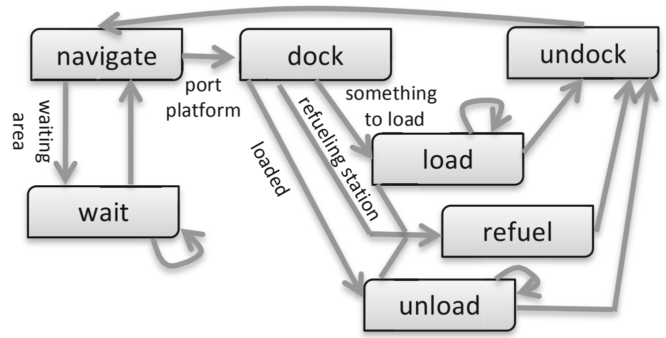

Fig. 1. Finite state automaton describing actions and allowed transitions (with some conditions) between the actions in the Petrobras challenge

Actions in the model have specific durations that are given by action parameters (such as locations and current load for the navigation actions). Hence, in the planning terminology we should rather talk about *planning operators* and actions are obtained by setting values of the parameters. The capacity of the vessel is modeled in the action, for example, the transition to a loading action is allowed only if there is enough capacity in the vessel.

So far we discussed plans for a single vessel, but if there are more vessels in the problem, their plans interact. For example at most two vessels can be docked at the port at the same time. We check these synchronization constraints when adding a new action to the plan as described in the next section.

3.2 Solving Approach

The solving algorithm uses the round-robin approach, where an action is selected for each vessel provided that the last action for that vessel finished before the rolling horizon. Figure 2 demonstrates the left-to-right round-robing solving approach that combines planning (action selection) with scheduling (allocation to time and resources). At the beginning all vessels are waiting so in the first step, we select an action for each vessel. There is only one exception of this process – if a vessel is waiting and a new waiting action is selected, we only prolong the existing waiting action for that vessel. Waiting action is the only action with arbitrary duration so it is possible to set its duration to any time. Action selection represents the choice point of the search algorithm. The only "heuristic" for action selection is the fixed order of actions in the model specification, where for example *unloading* is before *loading* which is before *refueling* for a docked vessel. There are also "control rules" encoded in the action descriptions – for example, the navigation action for an empty vessel goes only to a port with some remaining cargo, or to a refueling station, or to a waiting area.

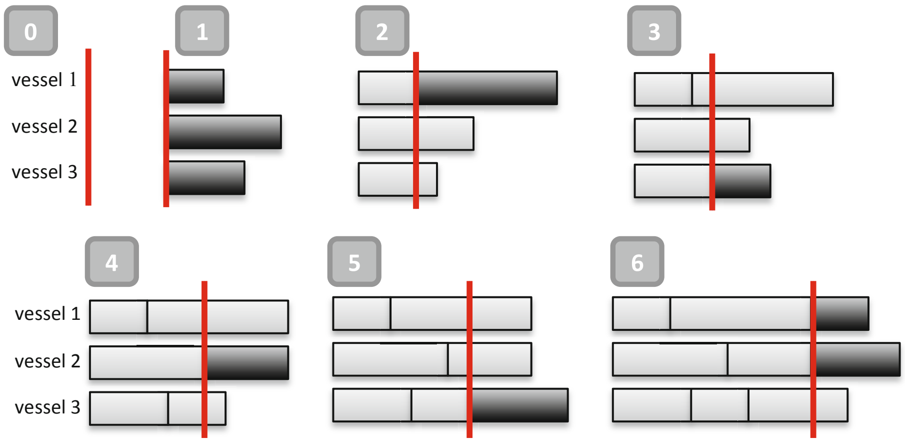

Fig. 2. Illustration of the left-to-right integrated planning and scheduling approach with a rolling horizon (vertical line). Newly added actions are displayed as black rectangles; x-axis represents time.

We have implemented the above solving approach in B-Prolog using tabling [1]. It means that we use depth-first search with remembering visited states – the state is represented by current states of vessels and a list of cargo items to be still delivered. In each step, we select an action for each vessel and move the time horizon. The core idea of this "planning" algorithm can be described using the following abstract code (this is actually an executable code in Picat [3] the follower of B-Prolog):

```
table (+,-,min)
plan(S,Plan,Cost),final(S) =>
    Plan=[],Cost=0.
plan(S,Plan,Cost) =>
    action(Action,S,S1,ActionCost),
    plan(S1,Plan1,Cost1),
    Plan = [Action|Plan1],
    Cost = Cost1+ActionCost.
```

As the reader can see, the search procedure is very simple. The real power of the solving approach is hidden in the action model and in the tabling mechanism. Tabling is important to save visited states so they are not re-explored. In the above code, for each state S the tabling mechanism stores the found Plan while minimizing the Cost of the plan (the cost is measured by makespan in the Petrobras problem). It is a form of a branch-and-bound procedure.

3.3 Meta-actions

Though the presented action model already included some sequencing restrictions, we have found experimentally that the model did not scale up well. In fact, it worked only for a single vessel with a few cargo items to deliver. By exploring the generated plans we noticed two types of erratic behavior. If more cargo items were available for delivery, all "free" vessels headed to the port, where cargo was located. This behavior was caused by preferring the navigation action to other actions if some cargo should be delivered. As the cargo was available before the first vessel loaded it, the other vessels "believed" that there is still some cargo to deliver and so transport to the port was planned for them. The second problematic behavior was that vessels left the waiting area just to refuel and then returned back to the waiting area.

Though the naïve model was not competitive to solve the problem, it showed the core ideas of our proposed modeling approach. The reasonable sequences of actions are modeled using a finite state automaton. To follow some transition a specific condition must be satisfied. We can make this model more efficient by grouping sequences of actions into a meta-action similarly to the MCTS approach [14]. Each meta-action may be a sequence of primitive actions with possible alternatives. We propose a model using four meta-actions with more specific conditions to apply the actions. Figure 3 shows the resulting finite state automaton including the transition conditions. Each meta-action decomposes into primitive actions while applying additional conditions on the actions. For example, the *Deliver* action starts with the *navigate* action but the destination is selected only from the destinations of loaded cargo items (a choice point). The next action in the sequence is *docking* followed by *unloading* and, if possible *refueling* done in parallel with *unloading*. We always unload all cargo items for a given destination and we always refuel the full tank (deterministic choice). The last action in the sequence is *undocking*. The *Deliver* action can only be used if some cargo is loaded to the vessel. Similarly, the *Pickup* action is applicable only if the vessel is empty and there is some cargo to deliver. We select the port where cargo is available (choice point) and pre-allocate some cargo items to the vessel (choice point). These conditions ensure that vessels are moving only when necessary. The *Pickup* action then decomposes to *navigate*, *docking*, *loading* and *refueling*, and *undocking* actions. Note that there could be more *loading* actions if more cargo items are loaded. As the order of loaded items is not important, only a single sequence of loading actions is explored during the decomposition (based on the fixed order of cargo items). This further reduces the search space – equivalent permutations of *loading* actions are not explored (similarly for *unloading* actions). The last two actions are *Waiting* and *Go2Wait* that are applicable if the vessel is empty and it is in the waiting area (then *Waiting*) or elsewhere (*Go2Wait*). The *Go2Wait* action decomposes to the *navigate* action, but if there would not be enough fuel then the vessel navigates to a refueling station before navigating to the waiting area. The three actions that include transport – *Pickup*, *Delivery*, *Go2Wait* – force the vessels to do only "reasonable" moves. If the vessel is empty, it can either go to a waiting area or to some port to pickup cargo, if any cargo is available. No other movement is allowed. Similarly, if the vessel is loaded, it can go only to some platform where some loaded cargo should be delivered.

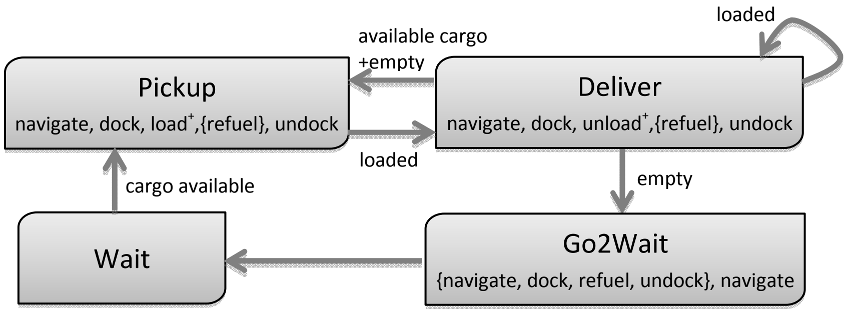
loaded
available cargo +empty
Pickup
navigate, dock, load$^+$,{refuel}, undock
Deliver
navigate, dock, unload$^+$,{refuel}, undock
loaded
cargo available
empty
Wait
Go2Wait
{navigate, dock, refuel, undock}, navigate

Fig. 3. Meta-actions and allowed transitions between meta-action in the Petrobras domain

The meta-actions allow users to specify expected sub-plans with conditions when to apply the sub-plans and non-determinism to be resolved by the planner (what cargo by which vessel). The user has better control about how the plans look like while leaving some decisions to the solver. The major caveat is some loss of flexibility. By specifying the sub-plans we may omit possible plans that were not assumed by the user. For example, in our model, we do not allow to pickup new cargo while some cargo is still loaded to the vessel. Also the cargo is only unloaded at its final destination. In particular, it is not possible to deliver the cargo "half-way" and using another vessel to deliver it to the final destination. These restrictions were intentional to reduce exploration of "unwanted" plans.

4 Experimental Results

To evaluate the proposed modeling and solving techniques we compare them with the best approaches from [14], namely the Filuta planner and the MCTS approach. We do not include SGPlan as its results were poor compared to other approaches. The Filuta planner minimized makespan identically to our method, while the MCTS planner used a manually tuned ad-hoc objective function that combined makespan, the number of actions, and fuel consumption: *usedFuel*+10**numActions*+5**makespan*. We re-use here the results reported in [14] where the experiments were run on the Ubuntu Linux machine equipped with Intel® Core™ i7-2600 CPU @ 3.40GHz and 4GB of memory and the planners were allowed to use approximately 10 minutes of runtime. Our method was implemented in B-Prolog, we run the experiments on the MacOS X 10.7.5 (Lion) machine with 1.8 GHz Intel® Core™ i7 CPU and 4GB of memory, and we report the best results found within one minute of runtime (we did not observe any further improvement when running B-Prolog longer).

We will first present the results for the original Petrobras problem described in [15], which consisted of 10 vessels and 15 cargo items to deliver, with the original fuel tank capacity for all vessels set to 600 liters. Table 1 shows the comparison of makespan and fuel consumption that were two major objectives in the Petrobras challenge. We can see that our approach is significantly better than the Filuta planner in both objectives and it also beats the best so-far approach based on MCTS in the makespan though the total fuel consumption is worse.

Table 1. Results for the Petrobras problem from [15]

Planner	*Makespan*	*Fuel*
Filuta	263 (1.62x)	1989 (2.24x)
MCTS	204 (1.26x)	**887** (1.00x)
B-Prolog	**162** (1.00x)	1263 (1.42x)

To compare the approaches in more detail, the paper [14] proposed several random benchmarks based on the Petrobras domain, where the number of cargo items to deliver and the number of available vessels varied. They also varied the fuel tank capacity, but according to the experiments this had a limited impact of performance so we kept the fuel tank capacity at 600 liters. We used two scenarios from [14] with 3 vessels and 10 vessels (called Group A and Group B in [14]), and we varied the number of cargo items from 1 to 15. Figure 4 shows the comparison of makespan for all three planners. We can observe that our approach is better for problems with the smaller number of cargo items, while the MCTS method takes lead when the number of cargo items increases. In fact, for the first six problems in each group, our system found (and proved) makespan-optimal plans, while the other two planners are sub-optimal only. We were also consistently better than the Filuta planner.

Regarding the fuel consumption (Figure 5), our planner is closer to the Filuta planner; we are mostly better but there were problems on which Filuta found plans with less fuel consumption. The MCTS method was consistently the best planner regarding fuel consumption, though this is not surprising as the other two planners optimized makespan only.

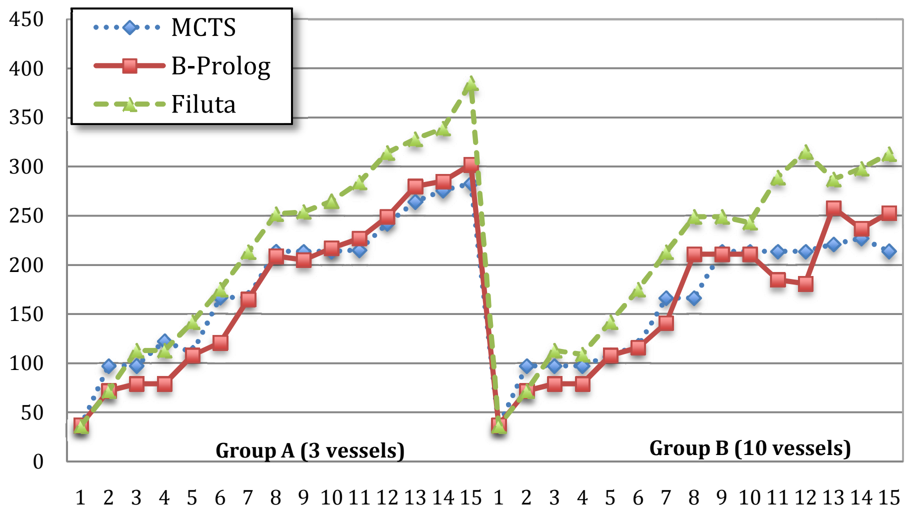

Fig. 4. Comparison of makespan for problems with different numbers of cargo items

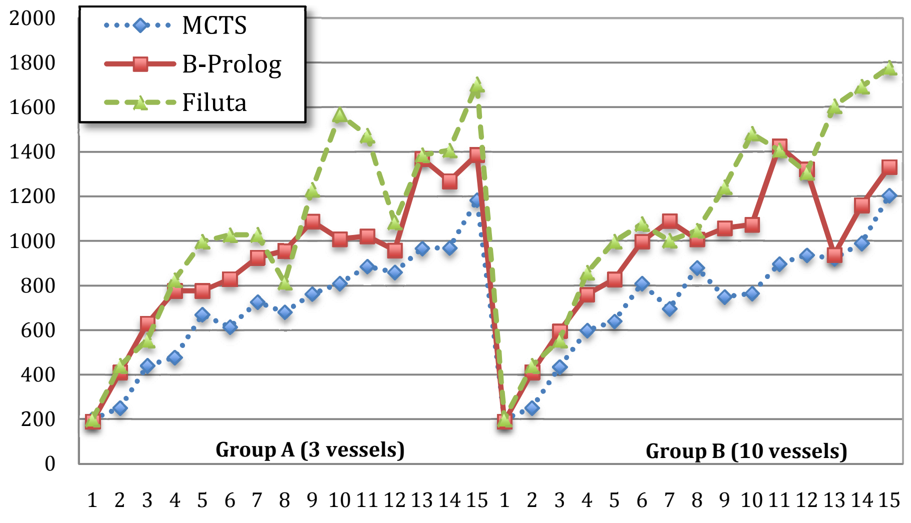

Fig. 5. Comparison of fuel consumption for problems with different numbers of cargo items

In the final experiment, we looked at the behavior of our planner when exploring the feasible plans. Recall that our method mimics the branch-and-bound algorithm in the sense that it starts with the first feasible plan and then outputs only better plans. Figure 6 shows the evolution of makespan and fuel consumption for the original Petrobras problem (so the order of solutions is from right to left as the makespan decreases). The figure shows a bit surprising “jumps” of fuel consumption when the makespan is decreasing. It may seem that this can be explained by increasing the number of vessels as the makespan decreases, but adding one more vessel to the plan does not always correspond with the increased fuel consumption. This behavior requires more detailed study that can also uncover why the plans generated by our method have larger fuel consumption than the MCTS method.

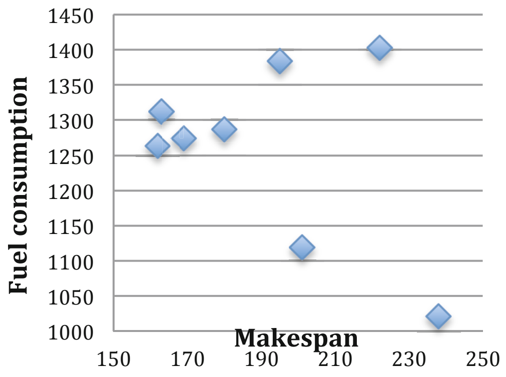

Fig. 6. Makespan and fuel consumption in plans found for the Petrobras challenge

5 Conclusions

We proposed a novel framework for describing planning problems that is based on meta-actions and transitions between them accompanied by conditions when the transitions can be used. Each meta-action decomposes into primitive actions; again specific conditions can be imposed on the parameters of these actions and also on their composition. The main motivation was to give users more control over action sequencing with two primary goals: obtaining more predictable plans and improving efficiency of planning. We demonstrated the modeling principles using a single domain – the Petrobras challenge – and using a single solving approach – B-Prolog with tabling. Though the solving technique is very simple, the proposed approach was shown to be competitive with the leading technique for the Petrobras problem in terms of plan quality (our technique was also faster).

The next step is to decouple the modeling approach from the solving mechanism and applying it to other planning problems. We sketched the modeling principles informally; a formal description in the form of a modeling language is necessary for more general applicability. The presented approach can also drive further research in the tabling methods applied to optimization problems. Other solving techniques may also be applied, we are not aware of any current planner supporting all three presented features: meta-actions, predefined action sequencing, and control rules. It would be interesting to study how these three modeling techniques contribute to plan efficiency. Our preliminary experiments showed that their combination helped to successfully solve the Petrobras challenge.

Acknowledgements. Roman Barták is supported by the Czech Science Foundation under the project KnowSched (P202-10-1188).

References

1. B-Prolog, `http://www.probp.com`
2. International Planning Competitions, `http://ipc.icaps-conference.org`
3. Picat, `http://www.picat-lang.org`
4. Barták, R.: On Constraint Models for Parallel Planning: The Novel Transition Scheme. In: Proceedings of SCAI 2011. Frontiers of Artificial Intelligence, vol. 227, pp. 50–59. IOS Press (2011)
5. Botea, A., Enzenberger, M., Muller, M., Schaeffer, J.: Macro-FF: Improving AI planning with automatically learned macro-operators. Journal of Artificial Intelligence Research 24, 581–621 (2005)
6. Chaslot, G., Bakkes, S., Szita, I., Spronck, P.: Monte-Carlo tree search: A new framework for game AI. In: Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference. The AAAI Press (2008)
7. Doherty, P., Kvarnström, J.: TALplanner: An empirical investigation of a temporal logic-based forward chaining planner. In: Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning (TIME 1999), pp. 47–54. IEEE Computer Society Press, Orlando (1999)
8. Dvořák, F., Barták, R.: Integrating time and resources into planning. In: Proceedings of ICTAI 2010, vol. 2, pp. 71–78. IEEE Computer Society (2010)
9. Erol, K., Hendler, J., Nau, D.: HTN Planning: Complexity and Expressivity. In: Proceedings of AAAI 1994, pp. 1123–1128 (1994)
10. Gerevini, A., Long, D.: BNF description of PDDL 3.0 (2005), `http://cs-www.cs.yale.edu/homes/dvm/papers/pddl-bnf.pdf`
11. Hsu, C., Wah, B.W.: The SGPlan planning system in IPC-6 (2008), `http://wah.cse.cuhk.edu.hk/wah/Wah/papers/C168/C168.pdf`
12. Kovacs, D.L.: BNF definition of PDDL 3.1 (2011), `http://www.plg.inf.uc3m.es/ipc2011-deterministic/OtherContributions?action=AttachFile&do=view&target=kovacs-pddl-3.1-2011.pdf`
13. Muscettola, N.: HSTS: Integrating Planning and Scheduling. In: Intelligent Scheduling. Morgan Kaufmann (1994)
14. 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)
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 Scheduling and Planning Applications Workshop, pp. 8–16 (2012)
16. Warren, D.S.: Memoing for Logic Programs. CACM 35(3), 93–111 (1992)
17. Zhou, N.-F., Dovier, A.: A Tabled Prolog Program for Solving Sokoban. In: Proceedings of the 26th Italian Conference on Computational Logic (CILC 2011), pp. 215–228 (2011)