2012 IEEE 24th International Conference on Tools with Artificial Intelligence

Three Approaches to Solve the Petrobras Challenge

Exploiting Planning Techniques for Solving Real-Life Logistics Problems

Daniel Toropila[1,2],
[1]Computer Science Center
Charles University in Prague
Prague, Czech Republic
daniel.toropila@mff.cuni.cz

Filip Dvořák[2], Otakar Trunda[2], Martin Hanes[2], Roman Barták[2]
[2]Faculty of Mathematics and Physics
Charles University in Prague
Prague, Czech Republic
filip.dvorak@runbox.com, otatrunda@gmail.com, martin.hanes@gmail.com, roman.bartak@mff.cuni.cz

***Abstract*—The Petrobras domain is an abstraction of a real-life problem of resource-efficient transportation of goods from ports to petroleum platforms. Being a good example of a difficult problem standing on the borderline between planning and scheduling, this domain was proposed as a challenge problem at the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012). In this paper we describe three different ways of modeling and solving this domain: by utilizing classical planning, temporal planning, and finally, single-player games and Monte-Carlo Tree Search.**

Keywords – classical planning; temporal planning; domain modeling; randomized tree search; logistics domain

I. INTRODUCTION

Application of state-of-the-art techniques to real-world problems is one of the most difficult challenges of today's science and it is, also within the planning community, one of the strongest motivations driving the innovations in the field. The International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) focuses on bridging the gap between the real-world problems and existing planning techniques. In its forth edition in 2012 it proposed several challenge problems to be modeled and solved including the transportation domain based on a real-life problem from the Petrobras company.

In this paper we will first briefly introduce the Petrobras challenge and then we will describe three approaches to tackle this domain. Two of them are based on formulating the domain as a planning problem – we propose formulations for classical planning with some post-processing and for temporal planning. The third approach uses an ad-hoc solving technique based on Monte-Carlo Tree Search [1]. After that we will present the experimental comparison of all these approaches. The paper will be concluded by a summary of our contribution and suggesting possible further enhancements.

II. PROBLEM DESCRIPTION

The Petrobras challenge 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, the load influences vessels' speed and fuel consumption and the limited capacity of fuel tanks must also be assumed. Given a set of cargo items to deliver, the problem is to find a feasible plan that guarantees the delivery of the cargo items, respecting the constraints on the vessel and port 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 certain ports and platforms and must always have enough fuel to go to the nearest refueling station. The objective is to minimize a series of aspects of the problem such as the total amount of fuel used, the size of the waiting queue in the ports, the number of vessels used to solve the requests, the makespan, and the docking cost. A detailed description of the Petrobras domain can be found in [2].

III. TAKE 1: EXPLOITING CLASSICAL PLANNING

Solving temporal planning problems optimally is hard. Moreover, there are not many temporal planning systems available for use. However, for the cases when we are interested mostly in optimizing the used resources, such as consumed fuel, one of the possible approaches is to try solving the corresponding sequential planning problem (with numerical fluents) where the action durations are ignored, and where the amount of the used fuel is to be minimized. We are then not only solving a simpler problem, but there is also a plethora of successful sequential planning systems that we could employ for this purpose. Once having a solution for the simplified problem ready – a sequential plan – we can reincorporate back the action durations, concurrent resource constraints, and causal dependencies between the actions. Further, based on that information we can schedule the actions by assigning their start and finish timestamps, obtaining thus the required valid temporal plan.

Obviously, if our main objective was to minimize the total makespan, it would be quite naïve to completely ignore the time information during the optimization phase. However, some relation might exist between the plans that are fuel-efficient and the plans that are time-efficient. One of the goals of our work was also to study this relation – to see how much the makespan quality of the plan would degrade if we optimize solely the fuel consumption.

A. Sequential Petrobras Domain

In order to use some of the existing sequential planners our first task was to formally represent the problem described earlier as a planning domain (and its individual instances as

1082-3409/12 $26.00 © 2012 IEEE
DOI 10.1109/ICTAI.2012.34

```
(define (domain petrobras)
    (:requirements :typing :action-costs :fluents)

    (:types
        location vessel cargo - object
        logistics_location waiting-area - location
        platform port - logistics_location
    )

    (:predicates
        ;; location is waiting area
        (is-waiting-area ?loc - location)

        ;; vessel is at location (platform, port, waiting area)
        (at ?vessel - vessel ?where - location)

        ;; vessel is docked in port or platform
        (is-docked ?v - vessel ?where - location)

        ;; platform is able to provide refuel for vessel
        (platform-can-refule ?platform - platform)

        ;; cargo is at location
        (cargo-at ?c - cargo ?loc)

        ;; helping predicates
        ;; vessel was once docked at this location
        ;; (navigate action negates this predicate)
        (vessel-once-docked-at-location ?v - vessel ?l - location)

    )

(:functions
    ;; the sum of fuel used by all the ships - to be minimized
    (total-fuel) - number

    ;; the free space in a vessel
    (vessel-free-capacity ?vessel - vessel) -  number

    ;; how many vessels can be docked in location
    (free-docks ?loc - location)  - number

    ;; max free capacity of a vessel
    (max-vessel-free-capacity) - number

    ;; current tank of a vessel
    (fuel-level ?vessel - vessel) - number

    ;; max fuel level (after refueling) independent on vessel
    (max-fuel-level) - number

    ;; the tank consumption - depending on the state of vessel
    (navigation-cost-empty ?from ?to - location) - number
    (navigation-cost-nonempty ?from ?to - location) - number
    (navigation-cost-empty-to-nearest-refuel-loc
     ?waiting-area - location) - number
    (navigation-cost-nonempty-to-nearest-refuel-loc
     ?waiting-area - location) - number

    ;; the distance between locations
    (distance ?from ?to - location) - number

    ;; the weight of cargo
    (cargo-weight ?c - cargo) - number
)
```

Figure 1. Portion of the Petrobras domain represented in PDDL (requirements, types, predicates and functions).

planning problems). For this purpose we chose the PDDL language [3], as it is the most commonly supported input definition language of the modern planning systems and de-facto the industry standard. We use the PDDL version 3.0.

As mentioned above, we completely ignore the temporal information about the actions. What we however need to model and encode are the resource constraints, such as fuel consumption of the vessels, limited capacity of the vessels for loading the cargo and, finally, limited capacity of the ports to dock the vessels.

Figure 1 depicts the requirements, types, predicates and functions used in the PDDL representation of the Petrobras domain. The following actions are then constructed using the elements from Figure 1 (the complete definition of actions is omitted due to its length):

- *navigate-empty-vessel*
- *navigate-nonempty-vessel*
- *load-cargo*
- *unload-cargo*
- *refuel-vessel-platform*
- *refuel-vessel-port*
- *dock-vessel*
- *undock-vessel*

A complete definition of all actions is omitted due to its length; however, one can intuitively construct the above actions based on the names of the elements from Figure 1 and taking into account the following design decisions:

1) A vessel is allowed to navigate to a destination only if it has sufficient amount of fuel. Also, if the destination is a waiting area, the vessel must have enough fuel to reach the nearest refuel location from the target waiting area.

2) To load and unload a cargo, the vessel must be both at the location and docked.

3) The refuel operations always take the full tanks. The main reason for introducing this simplification is that the real-life refuel operation is short when compared to the loading or unloading operations. This can obviously cause the unnecessary fuel to be carried by a vessel, which, in case of an accident, could potentially increase a risk of ecological disaster. However, the unnecessary excess of fuel can be removed by simple post-processing of the generated plans.

4) The predicate vessel-once-docked-at-location has been introduced in order to prevent successive docking and undocking operations by the same vessel in the same port. The docking operation has the negation of this predicate among its preconditions and makes this predicate valid, while only the navigate actions can invalidate this predicate.

B. Choosing a Planner

The PDDL representation of the planning problem allows (in principle) using any available planning system capable of solving sequential planning problems with numerical fluents. Unfortunately, during our work we were able to successfully run and compute plans for the problems from the Petrobras domain using a single planner only, namely SGPlan 6.0 [4].

C. Building a Temporal Plan

After representing a given instance of the Petrobras domain using PDDL and then solving it using the SGPlan planner while optimizing solely the fuel consumption we obtain a sub-optimal sequential plan. However, for the purpose of the real-world execution we need to create a schedule of the plan or, in other words, to build a temporal plan where the duration of the actions is not ignored and the actions, if possible, are executed in parallel. This should significantly decrease the makespan of the final solution.

In order to assign the actions their start times we need to take into consideration the causal relations between the actions (i.e., which actions provide preconditions of a given action), the duration of the actions, and the concurrent resource constraints (e.g., how many ships can be simultaneously docked at any given time in a port). Also, for each action *a* providing precondition *p* for action *b* we need to make sure that no other action that would destroy the precondition *p* is executed between the termination of action *a* and the start of action *b*. The actions that provide preconditions for a given action we shall refer to as *supporting actions*.

We start with a sequential plan that was produced by the planner and for all actions we compute their durations based on the information from the Petrobras domain as defined originally. Then we scan the sequential plan from the first action to the last and for each action we assign its start time equal to the end time of its latest finishing supporting action. This way we obtain a parallel temporal plan that might, however, still contain two types of flaws:

1) threats that some action in the plan destroys a supporting link (causal relation) between a pair of actions; and

2) resource constraints conflicts caused by the concurrent use of the limited resources.

In order to remove these flaws we perform the following further modifications of the schedule. Given a threat where action *c* is executed between actions *a* and *b* (where action *a* provides precondition *p* for action *b*; action *c* destroys *p*) we have two options to remove the threat. Either the execution of action *a* is postponed after termination of action *c*, or the execution of action *c* is postponed after the termination of action *b*. Taking into consideration all threats in the plan would be an exponential combinatorial search to find out the resolution of all threats that would lead to the optimal parallel schedule of a given sequential plan. However, if we give up such optimality (the input sequential plan is also sub-optimal) we can use for the resolution the ordering of actions from the sequential plan, as we can be sure that in the sequential plan action *c* must have been placed either before action *a* or after action *b*. Our temporal plan will then respect such ordering to resolve the threats of causal dependencies.

Resolving the conflicts of concurrent use of a limited resources works similarly. Whenever a conflict is detected, the execution of a conflicting action is postponed to the earliest time when the resource becomes available.

Obviously, shifting the start time of an action also shifts the start time of all depending actions, creating potentially another threats and resource conflicts. We repeat this procedure until all the flaws are resolved. The termination of this procedure is guaranteed by the fact that in the worst case we will end up with the original sequential plan.

IV. Take 2: Exploiting Temporal Planning

In this section we describe an approach that is built upon a planning system with explicit time and resources. The modeling and handling of the explicit time is one of the key aspects in capturing the real-world problems where time plays a critical role. Explicit time allows us to both express concurrency between actions and optimize how the actions will actually be scheduled in time during the planning process. Since by the addition of explicit time we are already moving into the field of scheduling, we can as well make use the concept of explicit resources in planning and adapt techniques that have been developed for resources from the scheduling perspective. In the following sections we first describe the modeling approach for the temporal version of the Petrobras domain, then we briefly touch the ideas of the planning system we have chosen and we describe how the planner models the resources that are present in the domain.

A. Temportal Petrobras Domain

We model the domain using the PDDL version 3.1 with fluents and durative actions [5]. The model is an extension of the sequential model where we reuse all the definitions that have been set up and extend the model by action durations and additional concurrency conditions that follow the definition of the domain by not allowing the vessels to load and unload cargo onto the same ship in parallel.

B. The Filuta Planning System

Filuta [6] is a planning system that operates with explicit time and models resources contained in the problem individually, using efficient solving techniques dependent on the real-world behavior of the modeled resource. The planner models the state of the world using the SAS+ representation [7] and adds the temporal annotations of the world state using a temporal database [7] for each state variable.

The planner's main optimization criterion is the makespan of the plan, however the planner sacrifices completeness and optimality in favor of the performance. The key idea for the performance boost is the division of the planning problem into sub-problems, where each of them contains only a single goal, resembling thus the original STRIPS algorithm for classical planning [7]. The planner search procedure proceeds by picking an open goal and extending the current plan by actions that achieve the goal. The search for the extension of the current plan is complete and optimal, where the optimization criteria are the global makespan and the sum of all the local makespans of evolutions of temporal databases. Although the optimal solutions of the sub-problems do not form a globally optimal solution, the produced plans exhibit comparatively high quality especially in the logistic domains and domains that contain large numbers of non-trivial resources [6], which motivated its application for the Petrobras domain.

C. Modeling Resources in Petrobras

While the often seen approach to handle resources in planning is to generalize their states into logical statements that can be evaluated in the same way as the planning literals, Filuta focuses on identifying the exact behavior of each resource in

the problem in addition with using a specific ad-hoc solver for the resource. The specific solvers are significantly more efficient and more informative then generalized resources, therefore the planner gains better lower bounds that in turn allow faster and more efficient pruning of the search space. The resources used in the Petrobras domain are the following:

- *Unary Resource (Single-capacity Reusable Resource)* corresponds to a single machine that can support only one process at any time. This is the simplest resource for which the solver only enforces time constraints that do not allow two processes to overlap. This resource is used to represent loading and unloading operations of the cargo for each ship.
- *Consumable Resource (Multi-capacity Replenishable Resource)* is generally a resource that can be both consumed and produced in the system, but either the consumption or the production is strictly absolute (the level of the resource is set by an assignment of the value) and the other is strictly relative (the level of the resource is set by an increment or decrement). This resource is used to represent the fuel tank in each ship.
- *Reservoir (Multi-capacity Replenishable Resource)* contains, in contrary to a consumable resource, only relative production or consumption events. We use this resource to represent the capacity of each vessel and the capacity of each location each ship can sail to. Note that a reservoir with the capacity of one unit is a unary resource, which is a preferred (more efficient) model.

V. TAKE 3: GAME-INSPIRED AD-HOC APPROACH

In this section we present an ad-hoc solution designed specifically for the Petrobras domain. We have decided to implement own search algorithm that would be better suited to this problem rather than use existing tools. Planning software usually deals with a feasibility problem (i.e., whether the plan exists or not). In this case, however, the feasibility problem is quite easy to solve, a simple greedy algorithm can find the plan, but the optimality is the real issue. We decided to use the Monte-Carlo Tree Search algorithm [1] that works natively with the evaluation function and uses it during the search. To use this algorithm the domain must meet several requirements, which we will discuss later.

A. Search Algorithm

We use the Single Player Monte-Carlo Tree Search (MCTS) algorithm for planning and scheduling. This algorithm has been used successfully in several single player games (puzzles) where a fitness function is involved (e.g. SameGame [8]). In order to use it for planning and scheduling we need to adjust the domain representation a little. These adjustments will be discussed in the next section. Now we briefly describe the algorithm.

MCTS is a state-space search algorithm. Suppose we have an initial state and a successor function that for every state returns a set of successor states. This function generates a tree – the initial state is a root. Suppose now that the tree is finite (it has no infinite branches) and that every leaf is evaluated. MCTS searches for the best-evaluated leaf. It generates the tree by iterative expansion of selected nodes. The generated tree is asymmetric; its shape is determined by the fitness function. The most promising nodes are expanded deeper. The nodes to expand are selected based on two criteria:

1) Expectation – the estimated value of the node (the expected best value of a leaf in the subtree); it supports the exploitation.

2) Urgency – the value that slowly increases every time the node is not selected and decreases rapidly when the node is selected; it supports the exploration.

The expected value of the node is determined by performing several random simulations. The simulation starts in a given node and then selects random edges until it reaches a leaf node. The value of simulation is defined as the value of the leaf node. The expectation of the node is then calculated as the average value of all simulations that passed through the node.

B. Domain Representation

We represent the search space as a tree, every node represents a word state and every edge corresponds to an action. The path in the tree from the root to a leaf node corresponds to a plan. Since the MCTS algorithm has been originally developed for games, this representation is based on a game-like view of the problem. The tree corresponds to a game tree, the state to a game position and edges to possible moves. Every finite state represents a (complete) plan and is evaluated according to the fitness function. The preconditions of each action correspond to game rules (which determine what moves are possible in a given position).

We want to use the search algorithm for both planning and scheduling, so we have to include time in our representation. When the planner adds an action to the plan the action will be scheduled immediately. The start time will be set as the earliest possible time according to availability of the sources, and the end time will be determined by duration of the action. In other words, when two actions share one or more sources the schedule will be determined by the order of these actions in the plan. The schedule is computed directly from the plan without performing any search.

Since the planner determines the order of actions, it does scheduling as well. Moreover, the information about the (expected) fitness value of every initial part of the plan (and the schedule) is known to the planner during search. Hence the search is driven to the most promising areas of the search space. That means the search procedure can recognize the promising initial parts of the plan (that lead to good schedules) and target the search to these areas. This differs fundamentally from the other approaches where planning and scheduling was strictly separated – such as the first approach described in this paper. The plan was found with no respect to the schedule and then this single plan was scheduled and evaluated. In this model all searched plans together with their initial parts are scheduled and evaluated and the search is driven by the evaluation function.

As mentioned above, the MCTS algorithm requires the tree to be finite because of the random simulations. If the tree contains an infinite branch then the simulation may run indefinitely. If we designed the actions as proposed in the original paper (load, unload, navigate, dock, undock, refuel) then the tree would be infinite. For example, the sequence

Dock, Undock, Dock, Undock, etc., is an infinite branch. There are two ways to tackle this problem.

1) Add a heuristic function to the simulations: Instead of being completely random the simulations would be driven by a heuristic function that would only select reasonable actions. Purposeless sequences of actions would still be present in the tree and therefore would have to be explored at least partially.

2) A different model of the actions: We may try to design the actions in a different way so that the infinite branches wouldn't even occur in the tree. Than there is no need for a heuristic function and random simulations would be sufficient. (Note that the heuristic function can still be added for better performance.)

We have chosen the second approach and designed actions that would not lead to infinite branches in the tree. To achieve this we have added another level of abstraction to the representation. We still use the original set of actions, but planning is performed on a higher level using the abstract actions. Every abstract action is then translated to a sequence of original actions. The abstract actions are:

- *Load(Ship, Cargo)*
- *Unload(Cargo)*
- *Refuel(Ship, Station)*
- *GoToWaitingArea(Ship)*

This abstract layer represents the "intentions" of the planner (i.e., "what to do"). The underlying layer consisting of the original actions, on the other hand, carries out the execution of the intention (i.e., "how to do that"). Depending on the current world state the auxiliary actions are added if needed. The process of translation is deterministic; no search is performed in the underlying layer.

For example, suppose that the planner added the abstract action *Unload(Cargo)* to the plan. The following information is obtained:

- The ship that the Cargo is on;
- The current location of the Ship; and
- The target location of the Cargo.

If the current location of the Ship is the same as the target location of the Cargo then only a single underlying action *Unload* is performed, otherwise the abstract action is translated to the following sequence of the original actions: *Undock*, *Navigate*, *Dock*, *Unload*. A similar principle is used also when other abstract actions are translated – the actions *Dock*, *Undock* and *Navigate* are added if necessary, but they are not included in the planning process.

C. Evaluation Function

Since the simulations evaluate the inner nodes of the tree, we don't have to evaluate the partial plans but only complete plans – the leaf nodes. There may be leaf nodes that represent the dead ends of the plan. Those correspond to the situations where no action is available but not all tasks have been completed, for example, some ships have not returned to the waiting areas and cannot do so because they do not have enough fuel. It is not clear how to evaluate these nodes. We could simply evaluate them by some huge values so that they would not be interesting for the search algorithm, however, doing so would devalue the rating of all the nodes on the path from the root to the "dead-end leaf". This is due to the fact that the expectation rating is computed as the average of all simulations passing through the node, and thus one huge value could bias the average considerably.

In order to solve this problem we make sure that the dead-end leafs do not occur in the tree. Luckily a simple look-ahead technique can provide this. The worst thing to happen is that some ship gets stuck in the platform without enough fuel to navigate anywhere else. To prevent this we assign every platform a number that prescribes the minimum amount of fuel that a ship must have upon arrival to the platform. The limit will be set so that the ship could always navigate to the nearest refuel station. Computing the limits can be done during the pre-processing phase and therefore it does not slow down the planning process itself.

Removal of dead-end leaves and the fact that the search tree is finite provide us with an important observation that all of the leaf nodes in the search tree represent valid plans. Therefore finally, in order to calculate the evaluation of a leaf node π (a plan) we use the following function *f*:

$$f(\pi) = usedFuel + 10 * countOfActions + 5 * makespan \quad (1)$$

The reason for including in (1) also the number of actions of a plan is following: because the final makespan is affected only by the latest action, and because of the parallel character of the action execution, the number of actions that terminate prior to the termination of the latest action does not affect the final makespan of a plan. For example, if the plan was using two vessels with the first vessel having the longest schedule, the second vessel would be allowed to perform unrestricted number of actions without affecting the final makespan, given they would terminate prior the end of the latest action of the first vessel. This would indeed lead to an undesired behavior, which we are trying to prevent. The multiplicative constants used in (1) are introduced to balance the effect of using different units.

VI. Experimental Evaluation

We implemented all three approaches described above in order to evaluate their performance for the specified problem. For our experiments we have generated 60 different planning problems where the number of cargo items, the number of vessels and the capacity of the vessels' fuel tank was varied. All other properties of the domain, such as the number of ports and platforms, their capacities and the distances between them were used as defined in the original domain description [2]. We created four groups of generated problems based on the combination of fuel tank capacity and the number of available vessels. For each group we generated 15 problems with the increasing number of cargo items from 1 to 15 with randomly generated locations and target platforms:

1) *Group A* – 3 vessels, fuel tank capacity 600 liters
2) *Group B* – 10 vessels, fuel tank capacity 600 liters
3) *Group C* – 10 vessels, fuel tank capacity 800 liters
4) *Group D* – 10 vessels, fuel tank capacity 1000 liters

Except of the 60 randomly generated problems we also encoded the sample problem defined in the original description of the domain, which consisted of 10 vessels and 15 cargo items to deliver (with the original fuel tank capacity set to 600 liters).

The main reason for creating multiple groups with varied fuel tank capacity of the vessels was that during the initial experiments we have observed improved performance of the SGPlan planner for the problem instances with the capacity raised from the original 600 liters to higher values.

The experiments were run on the Ubuntu Linux machine equipped with Intel® Core™ i7-2600 CPU @ 3.40GHz and 4GB of memory. The timeout for solving a single problem using the SGPlan planner was set to 2 hours, however the problems that were solved by SGPlan were either solved within two minutes, or the planner ran out of time. For the Filuta planner the timeout was set to 10 minutes, while for the ad-hoc MCTS approach the timeout was set to 80.000 iterations (expansions of a tree node) without any improvement of the best solution found. In most of the cases this corresponded to the timeout used for the Filuta planner. All computed solutions were then evaluated using the following optimization criteria, all of which were to be minimized:

1) Consumed fuel – the total amount of fuel consumed by the vessels in order to satisfy all of the transporation orders.

2) Number of vessels – the count of the vessels required to execute a plan.

3) Makespan – the total duration of the plan execution.

4) Docking cost – the total amount of the docking fees to be payed for the time that vessels spent in the port (while being docked) during the execution of the plan.

The results of the experiments are depicted in Figures 2 – 5. The first metric that we were interested in was the fuel consumption. The expectation here was that the sequential planner optimizing purely this criterion, SGPlan, would exhibit the best performance, however while the quality of solutions for the small problem instances was good, this approach did not scale well to bigger instances. In fact, for the problems with 7 cargo items or more the SGPlan did not return any solution. The increased fuel tank capacity of the vessels helped to solve more instances, however the returned plans had very bad quality with many unnecessary actions navigating the vessels between various locations. The plans returned by the Filuta planner were also using relatively large amounts of fuel. This behavior is however not surprising as Filuta optimizes only the makespan. The ad-hoc MCTS approach provided the best solutions, together with the ability to scale well.

An interesting observation is that the majority of the plans returned by SGPlan were using only a single vessel, which is the direct effect of the fuel-only optimization. Such plans, however, take very long time to execute in practice, as there is almost no possibility to execute the actions in parallel and obtain thus better makespan. On the other hand, both Filuta and the ad-hoc MCTS approach were using approximately the same number of vessels, which was even for the bigger problems limited by the capacity of the ports (one port can dock only two vessels at the time), together with the fact that the load/unload actions took always the majority of the plan execution time, as can be seen in Figure 6. The natural consequence of this result is that the port capacity directly implies the number of vessels that could be used for reaching maximum parallelization of the transportation in order to achieve the best possible makespan.

The examination of the durations of the plans confirms, without any surprise, that higher utilization of the vessels leads to a much better makespan performance. The poor performance of SGPlan was expected since the planner completely ignores the temporal information and only the fuel cost is optimized. The best results were again obtained by the ad-hoc MCTS approach, however the solutions computed by the Filuta planner remain very competitive, especially given the fact that Filuta is a domain-independent planner, while the ad-hoc MCTS approach includes reformulations that leverage from the specific domain structure.

The results also show that the docking costs of the plans returned by various planners are basically identical. This observation has a very straightforward explanation: Because the docking/undocking time is very short compared to the duration of the loading/unloading, and because only one cargo item can be loaded onto a vessel at any time, the docking cost is basically defined by the time required to load all the cargo items onto the vessels, which is in fact only affected by the total number of cargo items.

Table I provides an overview of the results for the sample problem defined in the original description of the domain. As can be seen, the best results were obtained by the ad-hoc MCST approach, while the Filuta planner remains competitive in the makespan criterion. See Figure 6 for a visualization of the plan found by the ad-hoc MCTS approach.

TABLE I. ORIGINAL SAMPLE PROBLEM RESULTS

Sample Problem	**Optimization Criteria**			
	Fuel (l)	***Vessels***	***Makespan (h)***	***Docking (R$)***
SGPlan	1226 (1.38x)	**2**	467.0 (2.29x)	315k (1.01x)
Filuta	1989 (2.24x)	5 (2.5x)	263.0 (1.29x)	333k (1.07x)
MCTS	**887**	4 (2.0x)	**203.5**	**311k**

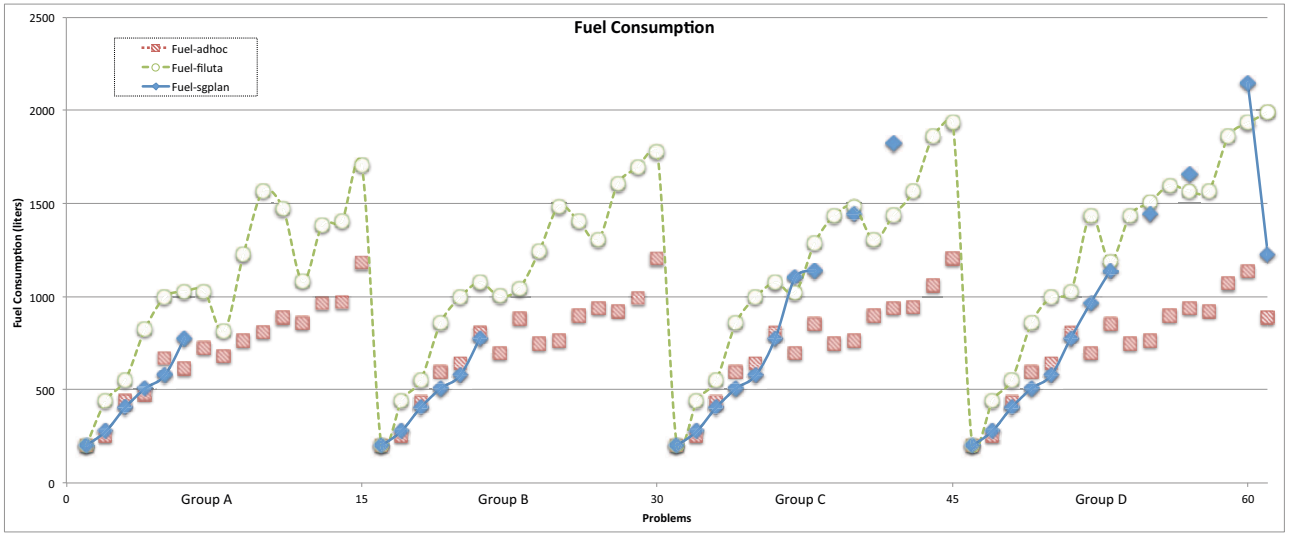

Figure 2. Fuel consumption of the plans for 60 generated testing problems and the original sample problem (depicted on the very right).

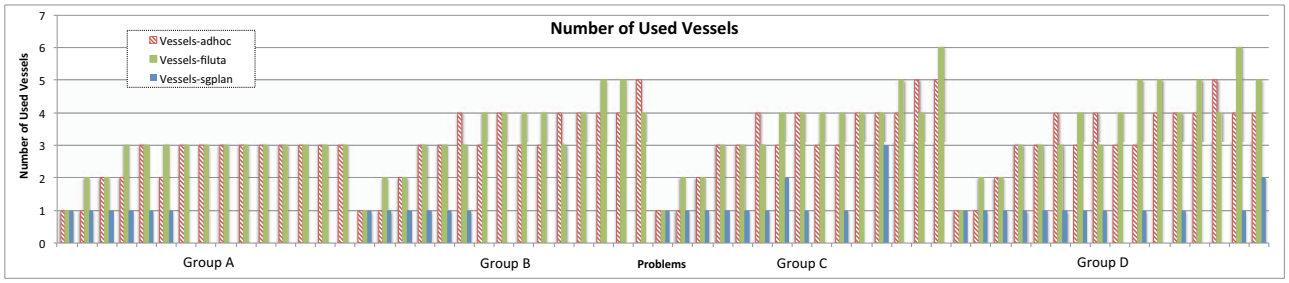

Figure 3. The number of used vessels of the plans for 60 generated testing problems and the original sample problem (depicted on the very right).

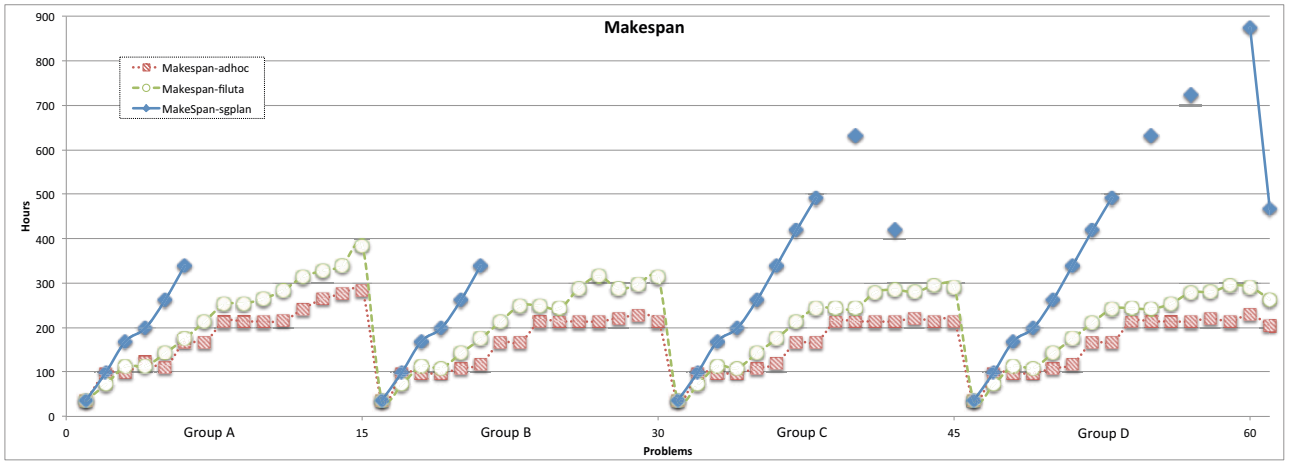

Figure 4. Makespan of the plans for 60 generated testing problems and the original sample problem (depicted on the very right).

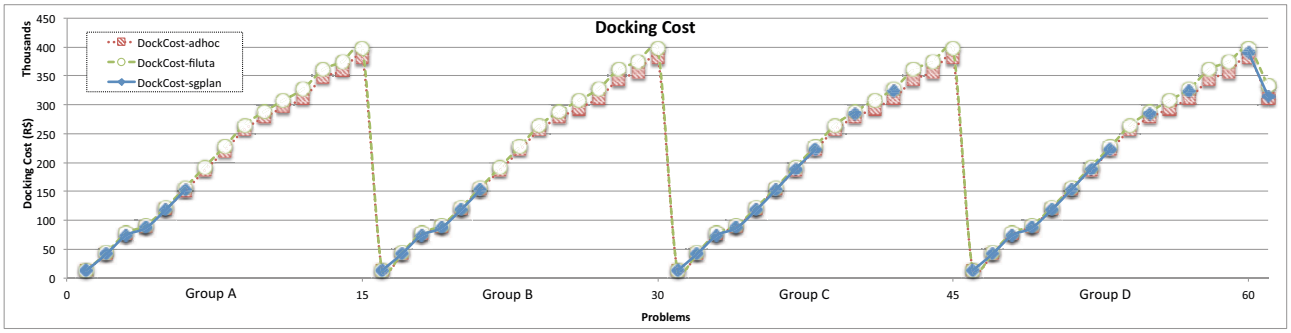

Figure 5. Docking cost of the plans for 60 generated testing problems and the original sample problem (depicted on the very right).

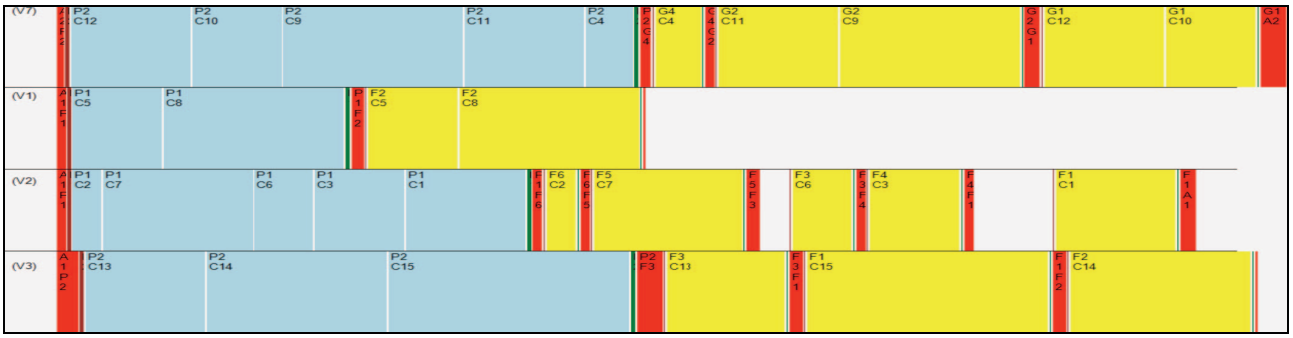

Figure 6. Visualization of the best plan found by the Ad-hoc MCTS approach. Each row corresponds to the schedule of a particular vessel.

VII. Conclusions and Future Work

In this paper we presented three approaches to solve the Petrobras challenge that was proposed for the fourth International Competition in Knowledge Engineering for Planning and Scheduling, ICKEPS 2012. First, we exploited the existing technology for classical sequential planning with numerical fluents, represented by the SGPlan 6.0 system, by ignoring the temporal aspects of the domain. This approach did not scale well and generated plans that utilized only a single vessel in most cases, which lead to bad makespan performance. The second approach exploited the temporal planner Filuta with strong pruning capabilities based on the resource constraints. This approach generated good plans regarding the makespan criterion and used more ships but also more fuel. The best overall results were achieved by an ad-hoc approach based on the Monte-Carlo Tree Search algorithm and some specific reformulations of the problem. An interesting observation is that the domain-independent planner Filuta is competitive with the ad-hoc approach regarding the makespan criterion.

In the future we plan to test solving both the sequential and temporal version of the domain also using other planners. For a sequential domain we would also like to construct a different optimization criterion that would include more vessels in the returned plans, improving thus indirectly the makespan. For the temporal planners, the integration of the fuel cost seems to be necessary in order to further improve the overall quality of the plans. Except of the studied optimization criteria, possible extensions are the ability to define priorities of the cargo items and minimization of the waiting queues in the ports.

Acknowledgements

The research is supported by the Czech Science Foundation under the contracts no. P103/10/1287, 201/09/H057, by the SVV project number 265 314, and by the Charles University Grant Agency under the contract no. 306011.

References

[1] G. Chaslot, S. Bakkes, I. Szita and P. Spronck, “Monte-Carlo tree search: A new framework for game AI,” Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, The AAAI Press, 2008.

[2] T. S. Vaquero, G. Costa, F. Tonidandel, H. Igreja, J. R. Silva and C. Beck, “Planning and scheduling ship operations on petroleum ports and platform,” Proceedings of the Scheduling and Planning Applications Workshop, 2012 pp. 8-16.

[3] A. Gerevini and D. Long, ”BNF description of PDDL 3.0,” 2005, http://cs-www.cs.yale.edu/homes/dvm/papers/pddl-bnf.pdf.

[4] C. Hsu and B. W. Wah, “The SGPlan planning system in IPC-6,” 2008, http://wah.cse.cuhk.edu.hk/wah/Wah/papers/C168/C168.pdf.

[5] D. L. Kovacs, “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.

[6] F. Dvořák and R. Barták, “Integrating time and resources into planning,” Proceedings of 22nd IEEE International Conference on Tools with Artificial Intelligence, Volume 2 (pp. 71-78). Arras, France, IEEE Computer Society, 2010, pp. 71-78.

[7] M. Ghallab, D. Nau, and P. Traverso, “Automated planning: Theory and practise,” Morgan Kaufmann Publishers, 2004.

[8] M. P. D. Schadd, M. H. M. Win, H. Jaap van den Herik and H. Aldewereld, “Addressing NP-complete puzzles with Monte-Carlo methods,” Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Volume 9, 2008.