6th IFAC Conference on Management and Control of Production and Logistics
The International Federation of Automatic Control
September 11-13, 2013. Fortaleza, Brazil

Toward Automated Planning Algorithms Applied to Production and Logistics

Sousa, A. R.*. Tavares, J. J. P. Z. S.*

** Manufacturing Automated Planning Lab, College of Mechanical Engineering, Federal University of Uberlândia, Av. João Naves de Ávila, 2121 - Uberlândia, Brazil (e-mail: alexandre_sousa@meca.ufu.br, jean.tavares@mecanica.ufu.br).*

Abstract: In recent years several studies have been made showing artificial intelligence techniques as enhancement proposals for real practical systems. One of such approaches is automated planning, in which knowledge of the system's behavior, expressed through a model, is used by a piece of software denominated automated planner to infer a sequence of actions capable of bringing the system from some initial state to an objective, a so called plan. To do such, the planner relies on some search algorithm capable of exploring the possibilities exposed by the model, and several different approaches have been used by different planners with varying degrees of success. This paper presents an insight on some of the most consolidated ones, both regarding deterministic and probabilistic domains, and focuses on search techniques and generic heuristics in order to assist the development of new algorithms focused on production and logistics. It also covers the main formal system modeling languages, such as STRIPS, PDDL and PPDDL, used by such planners.

1. INTRODUCTION

Automation is always related to new techniques and knowledge application. Nowadays, computer systems are mandatory to assist new automation projects. One of computer system field is artificial intelligence and automated planning is a special branch of it.

In recent years several studies have been made showing artificial intelligence techniques, such as automated planning, as enhancement proposals for real practical systems. The AI planning community is very committed to apply the developments already achieved in this area to real complex applications. However realistic planning problems bring great challenges not only for the designers during design processes but also for the automated planners during the planning process itself. (Sette et al., 2008)

More recently, some works have been developed focusing the practical applications for the automated planners using the itSIMPLE system (Vaquero, 2007). Several problems were already approached by that system, such as logistics problems in port systems (Dahal, 2003), the logistic inherent to the load and unloading processes of oil in São Sebastian's port (Sette *et al.*, 2008), the pumping of raw oil in pipelines (Li *et. al.*, 2005) and didactic initiatives (Tavares and Fonseca, 2011).

This work presents a review about automated planning modelling languages and algorithms, and it is a starting point for an ongoing research about integrating automated planning and the internal logistics of a manufacturing process.

This paper is structured as follows: first the main modelling languages are explained, then the classic planning algorithms, followed by a section about probabilistic planning and, finally, thoughts about applying these algorithms to logistic domains.

2. MODELING LANGUAGES

2.1 STRIPS

The formal language known as STRIPS actually borrowed its name from the original planner that used it, which is an acronym for Stanford Research Institute Problem Solver. STRIPS, the planner, is often cited as providing a seminal framework for attacking the "classical planning problem" (Fikes and Nilsson, 1993). In these classical planning problems, the world is defined as a static state that is only modified by a single agent that, through each action, brings the system from one state to the other. This simple-state problem formulation served as basis for automatic planning research during many years, and much of it was based specifically on the representation framework and reasoning methods developed in the STRIPS system.

The problem space for STRIPS is defined by the initial state *I*, the goal state *G*, and the operators *O* and their effects on the system model (Fikes and Nilsson, 1972). The planning problem can be expressed as follows:

$$P = (O, I, G) \tag{1}$$

The states are defined as arbitrary set of first-order predicate calculus well defined formulas (wffs). For example, to define a system where a box B sits at the location L, one could include the following wff:

$$at(B, L) \tag{2}$$

And then possibly include the following wff to state a box cannot be at two places at the same time:

$$\forall a,b,c\{[at(b,a) \wedge (a \neq c)] \Rightarrow \sim at(b,c)\} \tag{3}$$

The operators are defined by two main parts: one describing the preconditions necessary for execution, and one describing the effects. The definition of the effects of an operator is simply the lists of wffs the operator adds to the previous states, and the list of wffs it removes from said state (Fikes and Nilsson, 1972). In a more formal way, the action *o* can be defined as a set of a list of preconditions *pre*, a list of wffs to include in the new state *add* and the list of wffs to remove *del* as follows (Hoffmann and Nebel, 2001):

$$o = \left(pre(o), add(o), del(o)\right) \tag{4}$$

These operators can be grouped into families called schemata. Each operator schema can take in input parameters, upon which the member's operators are parametrized (Fikes and Nilsson, 1972). Extending upon the example mentioned earlier, assuming there is a robot to pick up boxes, an schema *pickup (posr,posb,b),* where *posr* is the current position of the robot, *posb* is the position of the box and *b* represents the box itself, can be defined instead of explicitly declaring the operators for every combination possible, thus somewhat simplifying the modelling process.

Overall, though, STRIPS, the planner and the language, are very limited both in planning issues it addresses and problems it can solve (Fikes and Nilsson, 1993). Even limited as it is, the STRIPS representation became one of the basis in automatic planning research for many years, most likely thanks to the severe simplifying assumptions it made, that allowed early progress to be made on the extreme difficulties of the general automatic planning problem.

2.2 PDDL

PDDL is an action-centred language, inspired by the well-known STRIPS formulations of planning problems. At its core is a simple standardisation of the syntax for expressing this familiar semantics of actions, using pre- and post-conditions to describe the applicability and effects of actions. The syntax is inspired by Lisp, so much of the structure of a domain description is a Lisp-like list of parenthesised expressions. (Fox and Long, 2003). The language, in its most basic and early versions, support the following features (McDermontt *et al.*, 1998):

- Basic STRIPS-style actions
- Conditional effects
- Universal quantification over dynamic universes (i.e., object creation and destruction),
- Domain axioms over stratified theories,
- Specification of safety constraints.
- Specification of hierarchical actions composed of subactions and subgoals.
- Management of multiple problems in multiple domains using differing subsets of language features (to support sharing of domains across different planners that handle varying levels of expressiveness).

An early design decision in the language was to separate the descriptions of parametrized actions that characterise domain behaviours from the description of specific objects, initial conditions and goals that characterise a problem instance. Thus, a planning problem is created by the pairing of a domain description with a problem description. The same domain description can be paired with many different problem descriptions to yield different planning problems in the same domain. (Fox and Long, 2003)

Further revisions added several features which enable the language to express more elaborate models. Among the most important additions are:

- Numeric expressions,
- Durative actions,
- Alternative objective functions (metric) (Fox and Long, 2003),
- Strong and soft problem goals – goals that must be achieved or are just desirable, respectively
- Strong and soft constraints on plan trajectories – similarly to the goals, strong ones have to be obeyed, whereas it is desired that soft ones are observed (Gerevini and Long, 2005).

The PDDL language is very modular, being factored into subsets of features, called requirements. Every domain defined using PDDL should declare which requirements it assumes. A planner that does not handle a given requirement can then skip over all definitions connected with a domain that declares that requirement, and won't even have to cope with its syntax.

To better illustrate the language, Figs. 1 and 2 exposes valid 2.1 PDDL codes that defines a simple domain and problem about logistics, taking fuel into account in a very primitive way. The fuel level is discretized into three levels (*full*, *half*, and *empty*), with each trip between two connected locations draining one level through the action *drive*. Two vehicles, the *car* and the *truck*, are declared, and each has a separate list of locations that are *accessible* from one another. The car begins at Paris with a full fuel tank and the truck starts with half a tank and at Rome. The goal of this planning problem is for the two vehicles to switch locations.

```
(define (domain vehicle)
  (:requirements :strips :typing)
  (:types vehicle location fuel-level)
  (:predicates (at ?v - vehicle ?p - location)
      (fuel ?v - vehicle ?f - fuel-level)
      (accessible ?v - vehicle ?p1 ?p2 - location)
      (next ?f1 ?f2 - fuel-level))
  (:action drive
    :parameters (?v - vehicle ?from ?to - location
                 ?fbefore ?fafter - fuel-level)
    :precondition (and (at ?v ?from)
                       (accessible ?v ?from ?to)
                       (fuel ?v ?fbefore)
                       (next ?fbefore ?fafter))
    :effect (and (not (at ?v ?from))
                 (at ?v ?to)
                 (not (fuel ?v ?fbefore))
                 (fuel ?v ?fafter)) ) )
```

Fig. 1. Transportation domain PDDL code (Fox and Long, 2003)

```
(define (problem vehicle-example)
  (:domain vehicle)
  (:objects
    truck car - vehicle
    full half empty - fuel-level
    Paris Berlin Rome Madrid - location)
  (:init
    (at truck Rome)
    (at car Paris)
    (fuel truck half)
    (fuel car full)
    (next full half)
    (next half empty)
    (accessible car Paris Berlin)
    (accessible car Berlin Rome)
    (accessible car Rome Madrid)
    (accessible truck Rome Paris)
    (accessible truck Rome Berlin)
    (accessible truck Berlin Paris)
  )
  (:goal (and (at truck Paris)
              (at car Rome))
)
```

Fig. 2. Transportation problem PDDL code (Fox and Long, 2003)

2.3 PPDDL

PPDDL1.0 is a first step towards a general language for describing probabilistic and decision theoretic planning problems, and is essentially a syntactic extension of PDDL 2.1. Note that, whereas the PDDL definition imposes a specific output plan structure for planners, PPDDL does not, except that only a single action can be executed at any point in time. The problem of plan representation has been left entirely to the planning systems, and planning systems may even choose to have no plan representation at all. (Younes and Littman, 2004).

PPDDL has been used in the probabilistic track at the International Planning Competition from the fourth to the sixth editions, and the latest instalment still provides automatic translations for PPDDL from the now used RDDL (Coles *et al.*, 2012).

The key extensions PPDDL brought is support for probabilistic effects (Younes and Littman, 2004) by the use of the syntax shown on Fig. 3 when declaring effects, where p_n represents the possibility of its associated sub-effect to be executed.

```
(...)
    :effect  (probabilistic p1 (effect)
                            p2 (effect)
                            (...)
                            pn (effect) )
(...)
```

Fig. 3. Definition of probabilistic effects

It is worth noting that a single PPDDL action schema can represent a large number of actions and a single predicate can represent a large number of state variables, meaning that PPDDL often can represent planning problems more succinctly than other representations. For example, the number of actions that can be represented using m objects and n action schemata with parity c is $m \cdot n \cdot c$, which is not delimited by any polynomial in the size of the original representation ($m+n$). Grounding is by no means a prerequisite for PPDDL planning, so planners could conceivably take advantage of the more compact representation by working directly with action schemata (Younes and Littman, 2004).

Markovian rewards, associated with state transitions, can be encoded using fluents. PPDDL reserves the fluent reward , accessed as (reward) or reward, to represent the total accumulated reward since the start of execution. Rewards are associated with state transitions through update rules in action effects.

PPDDL also makes adjustments to how goals should be interpreted. For regular probabilistic planning problems, the objective of the planner should be to maximize the probability of the stated goals to be achieved, which is stored in a special optimization metric defined as *goal-achieved*. For reward oriented planning problems, the default objective is to maximize the reward and the defined goals area set of absorbing states. There is also a special statement *(:goal-reward f)* that associates a one-time reward *f* is associated with entering a goal state (Younes and Littman, 2004).

2.4 RDDL

RDDL is a domain modelling language devised to model problem which pose a problem to the traditional (P)PDDL class of languages. Many important domains are difficult to express using (P)PDDL, like multi-elevator control with independent random arrivals, logistics domains with independently moving vehicles and noise, and UAVs with sensors for partially observed state (Sanner, 2010).

Instead of extending PPDDL, which is an extension to PDDL itself (Younes and Littman, 2004), an entirely new language was formulated since stochastic effects and concurrency are difficult to jointly reconcile in an effects-based language (Sanner, 2010).

Thanks to the expressiveness of the language, it has been adopted by the International Planning Competition in its seventh version, accompanied by a new variety of planning problems (Coles *et al.*, 2012).

A central design principle of RDDL is that the language should be simple and uniform with its expressive power deriving from composition of simple constructs. RDDL is based on the following principles:

- Everything is a parametrized variable (fluent or non-fluent),
- Flexible fluent types,
- The semantics is simply a ground Dynamic Bayes Net (DBN),
- General expressions in transition and reward functions,
- Classical Planning as well as General (PO)MDP – Partial Observed Markov Decision Process – objectives,
- State/action constraints.

Perhaps the most confusing issue for those familiar with PPDDL is the semantics of parametrized actions in RDDL. Each action fluent is a separate variable taking on a distinct value determined by the user. This is in contrast to the PPDDL view of actions where all of the action information is given in the action name and parameters. Here an action is not viewed as a parametrized variable so it does not make sense to say a PPDDL action consists of multiple ground variables as is the case in RDDL. The view of RDDL actions as templates for ground variables directly supports concurrency (Sanner, 2010).

3. CLASSIC PLANNING ALGORITHMS

3.1 Forward-chaining search

Forward-chaining search algorithms are very successful in the satisficing track of the International Planning Competition, so far as that LPG was the only winner in the tracks history to not be based on that (Coles *et al.*, 2012).

The search algorithms of the most prominent forward search planners, namely HSP (Bonnet and Geffner, 1998), FF (Hoffmann and Nebel, 2001), Fast Downward (Helmert, 2006), and LAMA-2008 and 2011 (Richter *et al.*, 2011), are guided by heuristics. These are estimates of the total cost of the solution that can be achieved by following a partial path, and are used so that the search algorithm can give priority to more promising paths, and are extracted automatically from planning domains through several techniques.

One of the consolidated techniques to extract heuristics from a model is to consider a relaxation of the problem into a simpler one (Bonnet and Geffner, 1998) by ignoring the *delete* list of STRIPS-based domains, an approach that was later adapted to work with numeric state variables as well (Hoffman, 2003). Then a solution to this relaxed problem is devised, be it by assuming independence between sub goals and calculating an additive heuristic (Bonnet and Geffner, 1998) or by applying a GRAPHPLAN (Blum and Furst, 1995) algorithm to solve the relaxed plan and use the calculated costs of the solution as an estimative (Hoffmann and Nebel, 1998). The FF planner goes one step further, and not only uses the estimated costs from the relaxed solution, but also extracts a list of *helpful actions* in order to decrease the algorithm's search space.

Later algorithms based on the Fast Downward planner implement additional heuristics through the generation of causal graphs. Informally, the causal graph contains an arc from a source variable to a target variable if changes in the value of the target variable can depend on the value of the source variable. Such arcs are included also if this dependency is of the form of an effect on the source variable. The planner also doesn't compute the heuristic estimate for each generated state, it rather computes them only for closed nodes, while computation is deferred for nodes on the search frontier (Helmert, 2006).

The Fast Downward planner also features a translation stage that transforms propositional tasks to multi-valued ones, as these have better structured causal graphs (Helmert, 2006). This concept was inherited by the planner LAMA (Richter *et al.*, 2011).

LAMA also implements its own scheme of heuristics by using the concept of *landmarks* in a *multi-heuristic search* context grouped with the original FF estimates. These landmarks are variable assignments that must occur at some point in every solution plan, and they are obtained by backchaining already known landmarks, starting with the goals (that are landmarks by definition).

HSP and FF are based on variants of the *hill climbing* algorithm, where the best node is expanded and all other ones are discarded. While time-efficient, it can get stuck in local minima. The HSP planner approaches this issue by restarting the search if needed (Bonnet and Geffner, 1998), whereas FF deals with it by discarding the hill climbing algorithm altogether and resorting to a *greedy best-first* search (Hoffmann and Nebel, 2001), which is also used, with adaptations, for Fast Downwards (Helmert, 2006). LAMA also uses greedy best-first search to find an initial solution, but builds upon that by running a series of weighted A* searches with decreased weight using the by-then best known solution for pruning the search (Richter *et al.*, 2011).

3.2 LPG

LPG is the winner of the third International Planning Competition, and is based on stochastic local search (Coles *et al.*, 2012). It works basically by generating an initial *planning graph* and iteratively modifying it until a solution graph is obtained.

A planning graph is a directed acyclic levelled graph with two kinds of nodes and three kinds of edges. The levels alternate between a fact level, containing fact nodes, and an action level containing action nodes. An action node at level t presents an action (instantiated operator) that can be planned at time step t. A fact node represents a proposition corresponding to precondition of one or more actions at time step t, or to an effect of one or more actions at time step $t-1$. The fact nodes of level 0 represent the positive facts of the initial state of the planning problem (every fact that is not mentioned in the initial state is assumed false). There is also a special action a_{end}, whose preconditions are the goal fact nodes, and it represents the last action in any valid plan (Gerevini and Serina, 2002).

To better understand the workings of LPG one also needs to define *action graphs*. An action graph (A-graph) A of G is a subgraph of G containing a_{end} and such that, if a is an action node of in A, then also the fact nodes of G corresponding to the preconditions and positive effects of $[a]$ are in A, together with the edges connecting them to a.

The modifications each search step applies to a planning graph consist of adding a new action node to the current A-graph, or removing an action node from it (together with the relevant edges) in order to solve a randomly chosen constraint violation, also called inconsistencies, which represents either conflicting actions happening on the same level or actions lacking supporting preconditions. Which of the two actions are executed is chosen based on a special

heuristic that estimates the cost to support the chosen violation (Gerevini and Serina, 2002).

3.3 SGPLAN

SGPlan is different from the planners discussed above in that it is not a distinct planning algorithm, but rather a partitioning framework that calls other algorithms to solve a batch of ordered subproblems which are, in theory, faster to solve. (Chen *et al.*, 2004).

The planner works on two levels. In the global level, we select a suitable order for the planner to solve the partitioned subgoals, introduce artificial global constraints to enforce that the solution of one subgoal solved later does not invalidate that of an earlier subgoal, and resolve violated global constraints using the theory of extended saddle points. In the local level, we perform a hierarchical decomposition of first-level.

The ordering in the global level is done through three heuristics. The first one is *reasonable ordering*, where if a subgoal can't be achieved without invalidating another, it must come before it. Failing that there is *irrelevance ordering*, where between two subgoals, the one with less irrelevant actions is solved first. In case the first two levels can't order two subgoals, *precondition ordering* is applied, giving priority to the one with the larger minimum number of preconditions of supporting actions. Pairs of actions not ordered in any level are sorted randomly.

In the local level, with a given subgoal *G* after first-level partitioning, SGPlan identifies intermediate second-level subgoals (or facts) that must be true in any plan that achieves *G* from a given initial state. These facts allow the construction of an intermediate goal agenda (IGA), which is an ordered list of agenda entries, each containing a set of intermediate facts.
Once the tasks are divided and ordered, they are handed to an actual planning algorithm. The implementation of SGPlan that won the Fourth International Planning Competition uses a modified implementation of Metric-FF and, when it fails, invokes LPG (Chen *et al.*, 2004).

4. PROBABILISTIC PLANNING ALGORITHMS

4.1 FF-REPLAN

FF-Replan is an action selection algorithm for online planning in probabilistic domains. FF-Replan has a very simple architecture. Given a new probabilistic planning problem, consisting of a domain description, a goal, and initial state, FF-Replan first generates a deterministic planning problem, removing all probabilistic information, then uses the deterministic planner FF (Hoffmann and Nebel, 2001) to compute a totally ordered plan for the generated deterministic problem. During execution of the resulting plan, if confronted with an unexpected state, the process is repeated with the unexpected state as the initial state, until a goal state is reached. Note that the determinization process is conducted once before execution begins and there is a potential improvement of the system by considering adaptive determinization (Yoon *et al.*, 2007).

Internally, FF-Replan maintains a partial state-action mapping using a hash-table which is initially empty. When FF-Replan encounters a state that is not in the table, then it determinizes the problem and synthesizes a plan using FF. FF-Replan then simulates the plan according to the deterministic action definitions resulting in a state-action sequence whose pairs are put in the hash table. The first action of the plan is then executed in the environment, which returns a new current state. FF-Replan thus produces a partial policy in an online fashion. Of course due to the deterministic approximation, the partial policy has no quality guarantees in general (Yoon *et al.*, 2007).

4.2 PROST

The PROST planning system is based on the upper confidence bounds applied to trees (UCT) algorithm (Kocsis and Szepesvári, 2006), a state-of-the-art approach for many problems of acting under uncertainty. As an *anytime* algorithm, UCT returns a non-random decision whenever queried, and terminates based on a timeout given to the system as a parameter. In the given time, UCT performs rollouts, where, as usual in Monte-Carlo approaches, outcomes of actions are sampled according to their probability (Keller and Eyerich, 2012).

The planner uses the detection of *reward locks*. Reward locks are states where where, no matter which action is applied and which outcome occurs, we end up in a state where we receive the same reward as before, and which is also a reward lock. These states can be goals or dead ends, and discerning between them is difficult, and PROST merely gives them higher priority during searches which effectively serves both to guide the search towards goal states and to avoid dead ends (Keller and Eyerich, 2012).

Another important aspect of PROST is how it exploits the structure of RDDL during the search process. For chance nodes, which are nodes that have several successors for some given action, the successor is chosen by sampling the outcome according to its transition probability. As transition functions for variables are independent from each other, the number of outcomes and with it the number of successors of a chance node might be exponential in the number of variables. Since RDDL provides a separate transition probability for each variable, they can be applied sequentially, drastically reducing the branch factor of chance nodes (Keller and Eyerich, 2012).

5. APPLICATIONS ON LOGISTIC DOMAINS

This work represents a starting point towards applying automated planning algorithms to logistic systems in manufacturing processes. This requires, beforehand, good knowledge of both the system in question and the strength and weaknesses of current planners. Classical and probabilistic planning algorithms were reviewed. To create a new automated planning system focusing on production and logistics, first of all, there is a need of requirement analysis.

The system itself can be divided in several levels, in which an upper level's actions can be expanded to fully-fledged plans generated in a lower one. This not only simplifies the

modelling process but also ensures modifications to the model do not enforce replanning efforts for the whole manufacturing system.

ACKNOWLEDGEMENTS

Authors are grateful to UFU, FEMEC, CAPES, FAPEMIG, CNPQ, Prof. Dr. Ricardo Fortes de Miranda and fellow colleagues at MAPL.

REFERENCES

Bonnet, B., and H. Geffner (1998). HSP: Heuristic search planner. In AIPS-98 Planning Competition Pittsburg, PA.

Bonnet, B., and H. Geffner (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1), pages 5-33.

Blum, A. L., and M. L. Furst, (1995). Fast Planning Through Planning Graph Analysis. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1636-1642.

Chen, Y., C.W. Hsu, and B.W. Wah (2004). SGPlan: Subgoal partitioning and resolution in planning. *Edelkamp et al.(Edelkamp, Hoffmann, Littman, & Younes, 2004)*.

Coles, A., A. Coles, A.G. Olaya, S. Jiménez, C.L. López, S. Sanner, and S. Yoon (2012). A survey of the seventh international planning competition. *AI Magazine*, 33(1), 83-88.

Dahal, K., S. Galloway, G. Burt, J. Mcdonald and I. Hopkins (2003). Port System Simulation Facility with an Optimization Capability. *International Journal of Computational Intelligence and Applications*, **3**, pages 395-410.

Fikes, R.E. and N.J. Nilsson (1972). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence, 2*, **3**, pages 189-208.

Fikes, R.E. and N.J. Nilsson (1993). STRIPS, a retrospective. *Artificial intelligence 59*, pages 227-232.

Fox, M. and D. Long (2003). PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research 20*, pages 61-124

Gerevini, A. and D. Long (2005). Plan constraints and preferences in PDDL. Technical Report, Department of Electronics for Automation, University of Brescia, Italy.

Gerevini, A. and I. Serina (2002). LPG: A planner based on local search for planning graphs with action costs. In: *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, pages 12-22.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research 26*, pages 191–246.

Hoffmann, J. (2003). The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State *Journal of Artificial Intelligence Research 20*, 291-341.

Hoffmann J. and B. Nebel (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research 14*, pages 253-302

Keller, T. and P. Eyerich (2012). PROST: Probabilistic planning based on UCT. *22nd International Conference on Automated Planning and Scheduling*, pages 119-127

Kocsis, L., and C. Szepesvári (2006). Bandit Based Monte-Carlo Planning. In: *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 282–293.

Li, J., L. Wenkai, I.A. Karimi, and R. Srinivasan (2005). Robust and Efficient Algorithm for Optimizing Crude Oil Operations, In: *American Institute of Chemical Engineers Annual Meeting*.

McDermott, D., M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld and D. Wilkins (1998). PDDL - The Planning Domain Denition Language. *The AIPS-98 Planning Competition Comitee*.

Richter, S., M. Westphal, and M. Helmert (2011). LAMA 2008 and 2011 (planner abstract). In *The 2011 International Planning Competition*, Deterministic Part, pages 50–54.

Sanner, S. (2010). Relational dynamic influence diagram language (RDDL): Language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

Sette, F.M., T.S. Vaquero, S.W. Park and J.R. Silva (2008). Are Automated Planers up to Solve Real Problems?. In: *Proceedings of the 17th World Congress The International Federation of Automatic Control (IFAC'08)*, pages 15817-15824.

Tavares, J.J.P.Z.S. and J.P.S. Fonseca (2011). Supply Chain Didactic Testing Bench With Automated Planning Tool. In *Proceedings of 21st International Conference on Production Research*. Sttutgart, Germany.

Vaquero, T.S. (2007). ITSIMPLE: Ambiente integrado de análise de domínios de planejamento automático. 316 f. MsC. diss. University of São Paulo, São Paulo, Brazil.

Yoon, S., A. Fern, and R. Givan (2007). FF-Replan: A baseline for probabilistic planning. *17th International Conference on Automated Planning and Scheduling*, **7**, pages 352-359.

Younes, H.L.S., and M.L. Littman (2004). PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.