Heuristic Planning with Time and Resources

Patrik Haslum[1] and Héctor Geffner[2]

[1] Department of Computer Science, Linköping University, Sweden
pahas@ida.liu.se
[2] Departamento de Computación, Universidad Simón Bolívar, Venezuela
hector@usb.ve

Abstract We present an algorithm for planning with time and resources, based on heuristic search. The algorithm minimizes makespan using an admissible heuristic derived automatically from the problem instance. Estimators for resource consumption are derived in the same way. The goals are twofold: to show the flexibility of the heuristic search approach to planning and to develop a planner that combines expressivity and performance. Two main issues are the definition of *regression* in a temporal setting and the definition of the *heuristic* estimating completion time. A number of experiments are presented for assessing the performance of the resulting planner.

1 Introduction

Recently, heuristic state space search has been shown to be a good framework for developing different kinds of planning algorithms. It has been most successful in non-optimal sequential planning, *e.g.* HSP [4] and FF [12], but has been applied also to optimal and parallel planning [10].

We continue this thread of research by developing a domain-independent planning algorithm for domains with metric time and certain kinds of resources. The algorithm relies on regression search guided by a heuristic that estimates completion time and which is derived automatically from the problem representation. The algorithm minimizes the overall execution time of the plan, commonly known as the *makespan*.

A few effective domain-independent planners exhibit common features: TGP [24] and TPSys [7] handle actions with duration and optimize makespan, but not resources. RIPP [15] and GRT-R [22] handle resources, and are in this respect more expressive than our planner. *Sapa* [5] deals with both time and resources, but non-optimally.

Among planners that exceed our planner in expressivity, e.g. Zeno [20], IxTeT [9] and HSTS [19], none have reported significant domain-independent performance (Jonsson *et al.* [13] describe the need for sophisticated engineering of domain dependent search control for the HSTS planner). Many highly expressive planners, *e.g.* O-Plan [26], ASPEN [6] or TALplanner [17], are "knowledge intensive", relying on user-provided problem decompositions, evaluation functions or search constraints[1].

2 Action Model and Assumptions

The action model we use is propositional STRIPS with extensions for time and resources. As in GRAPHPLAN [3] and many other planners, the action set is enriched with a *no-op* for each atom p which has p as its only precondition and effect. Apart from having a variable duration, a no-op is viewed and treated like a regular action.

2.1 Time

When planning with time each action a has a duration, $dur(a) > 0$. We take the time domain to be $\mathbb{R}^+$. In most planning domains we could use the positive integers, but we have chosen the reals to highlight the fact that the algorithm does not depend on the existence of a least indivisible time unit. Like Smith and Weld [24], we make the following assumptions: For an action a executed over an interval $[t, t + dur(a)]$

(i) the preconditions $pre(a)$ must hold at t, and preconditions not deleted by a must hold throughout $[t, t + dur(a)]$ and
(ii) the effects $add(a)$ and $del(a)$ take place at some point in the interior of the interval and can be used only at the end point $t + dur(a)$.

Two actions, a and b, are *compatible* iff they can be safely executed in overlapping time intervals. The above assumptions lead to the following condition for compatibility: a and b are compatible iff for each atom $p \in pre(a) \cup add(a)$, $p \notin del(b)$ and vice versa (*i.e.* $p \in pre(b) \cup add(b)$ implies $p \notin del(a)$).

2.2 Resources

The planner handles two types of resources: *renewable* and *consumable*. Renewable resources are needed during the execution of an action but are not consumed (*e.g.* a machine). Consumable resources, on the other hand, are consumed or produced (*e.g.* fuel). All resources are treated as real valued quantities; the division into unary, discrete and continuous is determined by the way the resource is used. Formally, a planning problem is extended with sets R_P and C_P of renewable and consumable resource names. For each resource name $r \in R_P \cup C_P$, $avail(r)$ is the amount initially available and for each action a, $use(a, r)$ is the amount used or consumed by a.

[1] The distinction is sometimes hard to make. For instance, *parc*Plan [18] domain definitions appear to differ from plain STRIPS only in that negative effects of actions are modeled indirectly, by providing a set of constraints, instead of explicitly as "deletes". *parc*Plan has shown good performance in certain resource constrained domains, but domain definitions are not available for comparison.

3 Planning with Time

We describe first the algorithm for planning with time, not considering resources. In this case, a *plan* is a set of action instances with starting times such that no incompatible actions overlap in time, action preconditions hold over the required intervals and goals are achieved on completion. The cost of a plan is the total execution time, or *makespan*. We describe each component of the search scheme: the search space, the branching rule, the heuristic, and the search algorithm.

3.1 Search Space

Regression in the classical setting is a search in the space of "plan tails", *i.e.* partial plans that achieve the goals provided that the preconditions of the partial plans are met. A regression state, *i.e.* a set of atoms, *summarizes* the plan tail; if s is the state obtained by regressing the goal through the plan tail P' and P is a plan that achieves s from the initial state, then the concatenation of P and P' is a valid plan. A similar decomposition is exploited in the forward search for plans.

In a temporal setting, a set of atoms is no longer sufficient to summarize a plan tail or plan head. For example, the set s of all atoms made true by a plan head P at time t holds no information about the actions in P that have started but not finished before t. Then, if a plan tail P' maps s to a goal state, the combination of P and P' is not necessarily a valid plan. To make the decomposition valid, search states have to be extended with the actions under execution and their completion times. Thus, in a temporal setting states become pairs $s = (E, F)$, where E is a set of atoms and $F = \{(a_1, \delta_1), \ldots, (a_n, \delta_n)\}$ is a set of actions a_i with time increments δ_i.

An alternative representation for plans will be useful: instead of a set of time-stamped action instances, a plan is represented by a sequence $\langle (A_0, \delta_0), \ldots, (A_m, \delta_m) \rangle$ of action sets A_i and positive time increments δ_i. Actions in A_0 begin executing at time $t_0 = 0$ and actions in A_i, $i = 1 \ldots m$, at time $t_i = \sum_{0 \leqslant j < i} \delta_j$ (*i.e.* δ_i is the time to wait between the beginning of actions A_i and the beginning of actions A_{i+1}).

State Representation A search state $s = (E, F)$ is a pair consisting of a set of atoms E and a set of actions with corresponding time increments $F = \{(a_1, \delta_1), \ldots, (a_n, \delta_n)\}$, $0 < \delta_i \leq dur(a_i)$. A plan P *achieves* $s = (E, F)$ at time t if P makes all the atoms in E true at t and schedules the actions a_i at time $t - \delta_i$. The initial search state is $s_0 = (G_P, \emptyset)$, where G_P is the goal set of the planning problem. Final states are all $s = (E, \emptyset)$ such that $E \subseteq I_P$.

Branching Rule A successor to a state $s = (E, F)$ is constructed by selecting for each atom $p \in E$ an establisher (*i.e.* a regular action or no-op a with $p \in add(a)$), subject to the constraints that the selected actions are compatible with each other and with each action $b \in F$, and that at least one selected action is not a no-op. Let SE be the set of selected establishers and let

$$F_{\text{new}} = \{(a, dur(a)) \,|\, a \in SE\}.$$

The new state $s' = (E', F')$ is defined as the atoms E' that must be true and the actions F' that must be executing before the last action in $F \cup F_{\text{new}}$ begins. This will happen in a time increment δ_{adv}:

$$\delta_{adv} = \min\{\delta \,|\, (a, \delta) \in F \cup F_{\text{new}} \text{ and } a \text{ is not a } \textit{no-op}\}$$

where no-op actions are excluded from consideration since they have variable duration (the meaning of the action no-op(p) in s is that p has persisted in the *last* time slice). Setting the duration of no-ops in F_{new} equal to δ_{adv}, the state $s' = (E', F')$ that succeeds $s = (E, F)$ becomes

$$E' = \{pre(a) \,|\, (a, \delta_{adv}) \in F \cup F_{\text{new}}\}$$
$$F' = \{(a, \delta - \delta_{adv}) \,|\, (a, \delta) \in F \cup F_{\text{new}}, \delta > \delta_{adv}\}$$

The cost of the transition from s to s' is $c(s, s') = \delta_{adv}$ and the fragment of the plan tail that corresponds to the transition is

$$P(s, s') = (A, \delta_{adv}), \text{ where } A = \{a \,|\, (a, \delta_{adv}) \in F \cup F_{new}\}$$

The accumulated cost (plan tail) along a state-path is obtained by adding up (concatenating) the transition costs (plan fragments) along the path. The accumulated cost of a state is the minimum cost along all the paths leading to s. The evaluation function used in the search algorithm adds up this cost and the heuristic cost defined below.

Properties The branching rule is sound in the sense that it generates only valid plans, but it does not generate *all* valid plans. This is actually a desirable feature[2]. The rule is *optimality preserving* in the sense that it generates *some* optimal plan. This, along with soundness, is all that is needed for optimality (provided an admissible search algorithm and heuristic are used).

3.2 Heuristic

As in previous work [10], we derive an admissible heuristic by introducing approximations in the recursive formulation of the optimal cost function.

[2] The plans generated are such that a regular action is executing during any given time interval and no-ops begin only at the times that some regular action starts. This is due to the way the temporal increments δ_{adv} are defined. Completeness could be achieved by working on the rational time line and setting δ_{adv} to the *gcd* of all actions durations, but as mentioned above this is not needed for optimality.

For any state $s = (E, F)$, the optimal cost is $H^*(s) = t$ iff t is the least time t such that there is a plan P that achieves s at t. The optimal cost function, H^*, is the solution to the Bellman equation [2]:

$$H^*(s) = \begin{cases} 0 & \text{if } s \text{ is final} \\ \min_{s' \in R(s)} c(s, s') + H^*(s') \end{cases} \tag{1}$$

where $R(s)$ is the regression set of s, *i.e.* the set of states that can be constructed from s by the branching rule.

Approximations. Since equation (1) cannot be solved in practice, we derive a lower bound on H^* by considering some inequalities. First, since a plan that achieves the state $s = (E, F)$, for $F = \{(a_i, \delta_i)\}$, at time t must achieve the preconditions of the actions a_i at time $t - \delta_i$ and these must remain true until t, we have

$$H^*(E, F) \geqslant \max_{(a_k, \delta_k) \in F} H^*\Big(\bigcup_{(a_i, \delta_i) \in F,\, \delta_i \geqslant \delta_k} pre(a_i), \emptyset\Big) + \delta_k \tag{2}$$

$$H^*(E, F) \geqslant H^*\Big(E \cup \bigcup_{(a_i, \delta_i) \in F} pre(a_i), \emptyset\Big) \tag{3}$$

Second, since achieving a set of atoms E implies achieving each subset E' of E we also have

$$H^*(E, \emptyset) \geqslant \max_{E' \subseteq E, |E'| \leqslant m} H^*(E', \emptyset) \tag{4}$$

where m is any positive integer.

Temporal Heuristic $\boldsymbol{H_T^m}$. We define a lower bound H_T^m on the optimal function H^* by transforming the above inequalities into equalities. A family of admissible temporal heuristics H_T^m for arbitrary $m = 1, 2, \ldots$ is then defined by the equations

$$H_T^m(E, \emptyset) = 0 \;\text{ if } E \subseteq I_P \tag{5}$$

$$H_T^m(E, \emptyset) = \min_{s' \in R(s = (E, \emptyset))} c(s = (E, \emptyset), s') + H_T^m(s') \;\text{ if } |E| \leqslant m \tag{6}$$

$$H_T^m(E, \emptyset) = \max_{E' \subseteq E, |E'| \leqslant m} H_T^m(E', \emptyset) \;\text{ if } |E| > m \tag{7}$$

$$\begin{aligned} H_T^m(E, F) = \max\Big[&\max_{(a_k, \delta_k) \in F} H_T^m\Big(\bigcup_{(a_i, \delta_i) \in F, \delta_i \geqslant \delta_k} pre(a_i), \emptyset\Big) + \delta_k\;, \\ &H_T^m\Big(E \cup \bigcup_{(a_i, \delta_i) \in F} pre(a_i), \emptyset\Big)\Big] \end{aligned} \tag{8}$$

The relaxation is a result of the last two equations; the first two are also satisfied by the optimal cost function. Unfolding the right-hand side of equation (6) using (8), the first two equations define the function $H_T^m(E, F)$ completely for $F = \emptyset$

and $|E| \leq m$. From an implementation point of view, this means that for a fixed m, $H^m_T(E, \emptyset)$ can be solved and precomputed for all sets of atoms with $|E| \leq m$, and equations (7) and (8) used at run time to compute the heuristic value of arbitrary states. The precomputation is a simple variation of a shortest-path problem and its complexity is a low order polynomial in $|A|^m$, where $|A|$ is the number of atoms.

For a fixed m, equation (6) can be simplified because only a limited set of states can appear in the regression set. For example, for $m = 1$, the state s in (6) must have the form $s = (\{p\}, \emptyset)$ and the regression set $R(s)$ contain only states $s' = (pre(a), \emptyset)$ for actions a such that $p \in add(a)$. As a result, for $m = 1$, (6) becomes

$$H^1_T(\{p\}, \emptyset) = \min_{a:p \in add(a)} dur(a) + H^1_T(pre(a), \emptyset) \tag{9}$$

The corresponding equations for H^2_T are in [11].

3.3 Search Algorithm

Any admissible search algorithm, *e.g.* A*, IDA* or DFS branch-and-bound [16], can be used with the search scheme described above to find optimal solutions.

The planner uses IDA* with some standard enhancements (cycle checking and a transposition table) and an optimality preserving pruning rule explained below. The heuristic used is H^2_T, precomputed for sets of at most two atoms as described above.

Incremental Branching In the implementation of the branching scheme, the establishers in SE are not selected all at once. Instead, this set is constructed incrementally, one action at a time. After each action is added to the set, the cost of the resulting "partial" state is estimated so that dead-ends (states whose cost exceeds the bound) are detected early. A similar idea is used in GRAPHPLAN. In a temporal setting, things are a bit more complicated because no-ops have a duration (δ_{adv}) that is not fixed until the set of establishers is complete. Still, a lower bound on this duration can be derived from the regular actions selected so far and in the state being regressed.

Selecting the Atom to Regress The order in which atoms are regressed makes no difference for completeness, but does affect the size of the resulting search tree. We regress the atoms in order of decreasing "difficulty": the difficulty of an atom p is given by the estimate $H^2_T(\{p\}, \emptyset)$.

Right-Shift Pruning Rule In a temporal plan there are almost always some actions that can be shifted forward or backward in time without changing the plan's structure or makespan (*i.e.* there is some "slack"). A right-shifted plan is one in which such movable actions are scheduled as late as possible.

As mentioned above, it is not necessary to consider all valid plans in order to guarantee optimality. In the implemented planner, non-right-shifted plans are excluded by the following rule: If s' is a successor to $s = (E, F)$, an action a compatible with all actions in F may *not* be used to establish an atom in s' when all the atoms in E' that a adds have been obtained from s by no-ops. The reason is that a could have been used to support the same atoms in E, and thus could have been shifted to the right (delayed).

4 Planning with Resources

Next, we show how the planning algorithm deals with renewable and consumable resources.

4.1 Renewable Resources

Renewable resources limit the set of actions that can be executed concurrently and therefore need to enter the planning algorithm only in the branching rule. When regressing a state $s = (E, F)$, we must have that

$$\sum_{(a_i,\delta_i)\in F\cup F_{\mathrm{new}}} use(a_i, r) \leqslant avail(r) \tag{10}$$

for every renewable resource $r \in R_P$.

Heuristic The H_T^m heuristics remain admissible in the presence of renewable resources, but in order to get better lower bounds we exclude from the regression set any set of actions that violates a resource constraint. For unary resources (capacity 1) this heuristic is informative, but for multi-capacity resources it tends to be weak.

4.2 Consumable Resources

To ensure that resources are not over-consumed, a state s must contain the remaining amount of each consumable resource r. For the initial state, this is $rem(s_0, r) = avail(r)$, and for a state s' resulting from s

$$rem(s', r) = rem(s, r) - \sum_{(a_i,t_i)\in F_{\mathrm{new}}} use(a_i, r) \tag{11}$$

for each $r \in C_P$.

Heuristic The heuristics H^m_T remain admissible in the presence of consumable resources, but become less useful since they predict completion time but not conflicts due to overconsumption. If, however, consumable resources are restricted to be *monotonically decreasing* (*i.e.* consumed but not produced), then a state s can be pruned if the amount of any resource r needed to achieve s from the initial situation exceeds $rem(s, r)$, the amount remaining in s. The amount needed is estimated by a function $need^m(s, r)$ defined in a way analogous to the function $H^m_T(s)$ that estimates time. The planner implements $need^1(s, r)$.

Because resource consumption is treated separately from time, this solution is weak when the time and resources needed to achieve a goal interact in complex ways. The H^m_T estimator considers only the fastest way of achieving the goal regardless of resource cost, while the $need^m$ estimator considers the cheapest way to achieve the goal regardless of time (and other resources). To overcome this problem, the estimates of time and resources would have to be integrated, as in for example [22]. Integrated estimates could also be used to optimize some combination of time and resources, as opposed to time alone.

4.3 Maintenance Actions

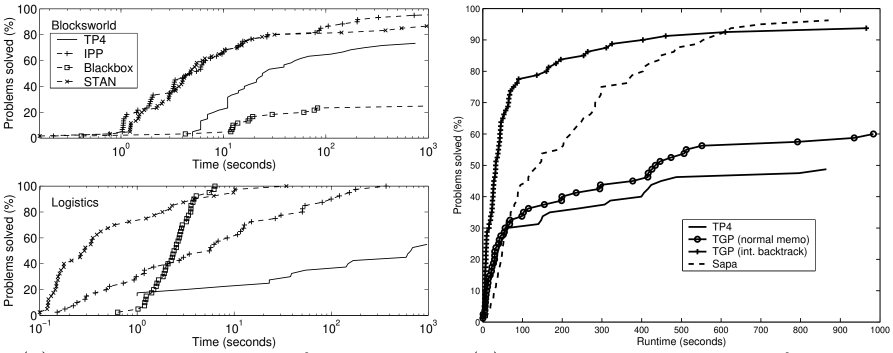
In planning, it is normally assumed that no explicit action is needed to maintain the truth of a fact once it has been established, but in many cases this assumption is not true. We refer to no-ops that consume resources as *maintenance actions.* Incorporating maintenance actions in the branching scheme outlined above is straightforward: For each atom p and each resource r, a quantity $use(\texttt{maintain}(p), r)$ can be provided as part of the domain definition and is set to 0 by default. Since the duration of a no-op is variable, we interpret $use(\texttt{maintain}(p), r)$ as the *rate* of consumption. For the rest, maintenance actions are treated as regular actions, and no other changes are needed in the planning algorithm.[3]

5 Experimental Results

We have implemented the algorithm for planning with time and resources described above, including maintenance actions but with the restriction that consumable resources are monotonically decreasing, in a planner called TP4[4]. The planner uses IDA* with some standard enhancements and the H^2_T heuristic. The resource consumption estimators consider only single atoms.

[3] This treatment of maintenance actions is not completely general. Recall that the branching rule does not generate *all* valid plans: in the presence of maintenance actions it may happen that some of the plans that are not generated demand less resources than the plans that are. When this happens, the algorithm may produce non-optimal plans or even fail to find a plan when one exists. This is a subtle issue that we will address in the future.

[4] TP4 is implemented in C. Planner, problems, problem generators and experiment scripts are available at `http://www.ida.liu.se/~pahas/hsps/`. Experiments were run on a Sun Ultra 10.



(a) Runtime distributions for TP4 and optimal non-temporal parallel planners on standard planning problems. A point $\langle x, y \rangle$ on the curve indicates that y percent of the problems were solved in x seconds or less. Note that the time axis is logarithmic.

(b) Runtime distributions for TP4, TGP and *Sapa* on problems from the simple temporal logistics domain. *Sapa*'s solution makespan tends to be around 1.25 – 2.75 times optimal.

Figure1.

5.1 Non-Temporal Planning

First, we compare TP4 to three optimal parallel planners, IPP, BLACKBOX and STAN, on standard planning problems without time or resources. The test set comprises 60 random problems from the 3-operator blocksworld domain, ranging in size from 10 to 12 blocks, and 40 random logistics problems with 5 – 6 deliveries. Blocksworld problems were generated using Slaney & Thiebaux's BWSTATES program [23].

Figure 1(a) presents the results in the form of runtime distributions. Clearly TP4 is not competitive with non-temporal planners, which is expected considering the overhead involved in handling time. Performance in the logistics domain, however, is very poor (*e.g.* TP4 solves less than 60% of the problems within 1000 seconds, while all other planners solve 90% within only 100 seconds), indicating that other causes are involved (most likely the branching rule, see below).

5.2 Temporal Planning

To test TP4 in a temporal planning domain, we make a small extension to the logistics domain[5], in which trucks are allowed to drive between cities as well as within and actions are assigned durations as follows:

[5] The goal in the logistics domain is to transport a number of packages between locations in different cities. Trucks are used for transports within a city and airplanes for transports between cities. The standard domain is available *e.g.* as part of the AIPS 2000 Competition set [1].

Actions	Duration	Actions	Duration
Load/Unload	1	Drive truck (between cities)	12
Drive truck (within city)	2	Fly airplane	3

This is a simple example of a domain where makespan-minimal plans tend to be different from minimal-step parallel plans.

For comparison, we ran also TGP and *Sapa*[6] The test set comprised 80 random problems with 4 – 5 deliveries. Results are in figure 1(b). We ran two versions of TGP, one using plain GRAPHPLAN-like memoization and the other minimal conflict set memoization and "intelligent backtracking" [14]. TP4 shows a behaviour similar to the plain version of TGP, though somewhat slower. As the top curve shows, the intelligent backtracking mechanism is very effective in this domain (this was indicated also in [14]).

5.3 Planning with Time and Resources

Finally, for a test domain involving both time and non-trivial resource constraints we have used a scheduling problem, called multi-mode resource constrained project scheduling (MRCPS) [25]. The problem is to schedule a set of tasks and to select for each task a mode of execution so as to minimize project makespan, subject to precedence constraints among the tasks and global resource constraints. For each task, each mode has different duration and resource requirements. Resources include renewable and (monotonically decreasing) consumable. Typically, modes represent different trade-offs between time and resource use, or between use of different resources. This makes finding optimal schedules very hard, even though the planning aspect of the problem is quite simple.

The test comprised sets of problems with 12, 14 and 16 tasks and approximately 550 instances in each (sets `J12`, `J14` and `J16` from [21]). A specialized scheduling algorithm solves all problems in the set, the hardest in just below 300 seconds [25]. TP4 solves 59%, 41% and 31%, respectively, within the same time limit.

6 Conclusions

We have developed an optimal, heuristic search planner that handles concurrent actions, time and resources, and minimizes makespan. The two main issues we have addressed are the formulation of an admissible heuristic estimating completion time and a branching scheme for actions with durations. In addition, the planner incorporates an admissible estimator for consumable resources that allows more of the search space to be avoided. Similar ideas can be used to optimize a combination of time and resources as opposed to time alone.

The planner achieves a tradeoff between performance and expressivity. While it is not competitive with either the best parallel planners or specialized schedulers, it accommodates problems that do not fit into either class. An approach

[6] We did not run *Sapa* ourselves. Results were provided by Minh B. Do, and were obtained using a different, but approximately equivalent, computer.

for improving performance that we plan to explore in the future is the combination of the lower bounds provided by the admissible heuristics H_T^m with a different branching scheme. See [8] for details.

Acknowledgments

We'd like to thank David Smith for much help with TGP, and Minh Binh Do for providing results for *Sapa*. This research has been supported by the Wallenberg Foundation and the ECSEL/ENSYM graduate study program.

References

1. AIPS competition, 2000. `http://www.cs.toronto.edu/aips2000/`.
2. R.E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.
3. A.L. Blum and M.L. Furst. Fast planning through graph analysis. *Artificial Intelligence*, 90(1–2):281 – 300, 1997.
4. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001. To appear.
5. M.B. Do and S. Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *Proc. 6th European Conference on Planning*, 2001.
6. A.S. Fukunaga, G. Rabideau, S. Chien, and D. Yan. ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proc. International Symposium on AI, Robotics and Automation in Space*, 1997.
7. A. Garrido, E. Onaindia, and F. Barber. Time-optimal planning in temporal problems. In *Proc. 6th European Conference on Planning*, 2001.
8. H. Geffner. Planning as branch and bound and its relation to constraint-based approches. `http://www.ldc.usb.ve/~hector`.
9. M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proc. 2nd International Conference on AI Planning Systems*, 1994.
10. P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling.* AAAI Press, 2000.
11. P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proc. IJCAI Workshop on Planning with Resources*, 2001. `http://www.ida.liu.se/~pahas/hsps/`.
12. J. Hoffman. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proc. 12th International Symposium on Methodologies for Intelligent Systems*, 2000.
13. A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in interplanetary space: Theory and practice. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*, 2000.
14. S. Kambhampati. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in Graphplan. *Journal of AI Research*, 12:1 – 34, 2000.
15. J. Koehler. Planning under resource constraints. In *Proc. 13th European Conference on Artificial Intelligence*, 1998.
16. R.E. Korf. Artificial intelligence search algorithms. In *Handbook of Algorithms and Theory of Computation*, chapter 36. CRC Press, 1999.

17. J. Kvarnstrom and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1):119 – 169, 2000.
18. J.M. Lever and B. Richards. *parc*PLAN: A planning architecture with parallel actions, resources and constraints. In *Proc. 9th International Symposium on Methodologies for Intelligent Systems*, 1994.
19. N. Muscettola. Integrating planning and scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan-Kaufmann, 1994.
20. J.S. Penberthy and D.S. Weld. Temporal planning with continous change. In *Proc. 12th National Conference on Artificial Intelligence (AAAI'94)*, 1994.
21. PSPLib: The project scheduling problem library. `http://www.bwl.uni-kiel.de/Prod/psplib/library.html`.
22. I. Refanidis and I. Vlahavas. Heuristic planning with resources. In *Proc. 14th European Conference on Artificial Intelligence*, 2000.
23. J. Slaney and S. Theibaux. Blocks world revisited. *Artificial Intelligence*, 125, 2001. See `http://arp.anu.edu.au:80/~jks/bw.html`.
24. D.E. Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. 16th International Joint Conference on Artificial Intelligence*, 1999.
25. A. Sprecher and A. Drexl. Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. I: Theory & II: Computation. Technical Report 385 & 386, Instituten für Betriebswirtschaftsrlehre der Universität Kiel, 1996.
26. A. Tate, B. Drabble, and J. Dalton. O-Plan: a knowledge-based planner and its application to logistics. In *Advanced Planning Technology*. AAAI Press, 1996.