Heuristic Planning with Time and Resources

Patrik Haslum[1] and Héctor Geffner[2]

[1] Department of Computer Science
Linköping University
S-58183 Linköping, Sweden
pahas@ida.liu.se
Fax: +46 13 28 44 99

[2] Departamento de Computación
Universidad Simón Bolívar
Aptdo 89000, Caracas 1080-A, Venezuela
hector@usb.ve
Fax: +58 212 906-3243

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. The two main issues are the definition of *regression* in the temporal setting and the definition of the *heuristic* for 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 [3] and FF [9], but has been applied also to optimal and parallel planning with good results [7].

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, known as *makespan*.

As far as we are aware, no effective domain-independent planner matches the expressivity of our planner, though a few exhibit some common features. For example, TGP [19] handles actions with duration and optimizes makespan, while RIPP [11] and GRT-R [15] handle resources, and are in this respect more expressive than our planner.

Among planners that exceed our planner in expressivity, e.g. Zeno [14], IxTeT [6] and HSTS [13], none have reported significant domain-independent performance (Jonsson *et al* [10] describe the need for sophisticated engineering of domain dependent search control for the HSTS planner). Planners like O-Plan [20] and ASPEN [4] are also very expressive but inherently domain-dependent.

2 Action Model and Assumptions

The action model we use is propositional STRIPS with extensions for time and resources. Like in GRAPHPLAN [2] 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 the fact that its duration is variable, a no-op is viewed and treated like regular actions.

2.1 Time

When planning with time each action 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 to employ reals to demonstrate that the algorithm does not depend on the existence of a least indivisible time unit. Like Smith and Weld [19], 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 $[t, t + dur(a)]$;
(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* refer to resources that 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 handled 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.

3 Example: Autonomous Spacecraft

Smith *et.al* [18] present a planning domain in which an autonomous spacecraft has to make a number of scientific observations in a limited time window and with limited resources, both renewable and consumable.

To make an observation, the spacecraft has to turn towards the target and the selected instrument (there may be several possibilities) has to be switched on and calibrated, which requires first turning towards a reference target. Turning operations take a variable amount of time. The spacecraft consumes fuel not only when it's turning, but also when maintaining a stable orientation. Likewise, power constraints limit the number of concurrently active instruments. This is of course a highly simplified description of spacecraft operations, but nevertheless presents a challenging planning problem.

A sample plan for achieving the goal `(HAVE_IMAGE A212 COLOR)` is[1]:

```
[0.00,22.00]  (TURN_TO NORTHSTAR EARTH)
[20.00,22.00] (SWITCH_ON CAM2)
[22.00,31.00] MAINTAIN(POWER_ON CAM2)
```

[1] Each action in the plan is preceded by its start and end time. For readability, maintenance actions have been coalesced and no-op's excluded.

```
[22.00,23.00] MAINTAIN(POINTING NORTHSTAR)
[22.00,23.00] (CALIBRATE CAM2 NORTHSTAR)
[23.00,30.00] (TURN_TO A212 NORTHSTAR)
[30.00,31.00] MAINTAIN(POINTING A212)
[30.00,31.00] (TAKE_IMAGE CAM2 A212)
```

A PDDL description of the domain and some experiments are given below.

4 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 provided that the initial state holds at time 0. 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.

4.1 Search Space

When doing regression in the classical setting, we search the space of "plan tails", *i.e.* partial plans that achieve the goals provided that the preconditions of the partial plans are met. A state in the regression *summarizes* the information in the plan tail. Indeed, 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' yields a valid plan. A similar decomposition is exploited in the forward search for plans. In either case, states are *sets of atoms*.

In the temporal setting, the situation is different as a set of atoms no longer summarizes a plan tail or plan head. E.g., the set s of all atoms made true by a plan head P at time t does not summarize the actions in P that have not completed before t. Thus, if plan tail P' maps s into a goal state, the combination of P and P' will not necessarily yield a valid plan. This decomposition becomes valid, however, once search states are extended with the actions under execution and the times needed for their completion. 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 and time increments δ_i.

In formalizing this idea, it will be useful to use an alternative representation for plans: Instead of sets of time-stamped actions $\{[t_0]a_0, \ldots, [t_n]a_n\}$, where t_i refers to the start time of action a_i, we represent plans in an equivalent form as sequences $\langle (A_0, \delta_0), \ldots, (A_m, \delta_m) \rangle$ of action sets A_i and positive time increments δ_i. In such plan, actions A_0 are executed at time $t_0 = 0$, and actions A_{i+1}, $0 \leq i < m$, are executed at time $t_{i+1} = \sum_{j=0,i} \delta_j$.

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 when 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 constraint that the selected actions are compatible with each other and with each action $b \in F$ (and that some action a is not a no-op). Let SE be the set of selected establishers, and let

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

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

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

where no-op actions are excluded from consideration as they have a 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-op's in F_{new} to δ_{adv}, the state $s' = (E', F')$ that succeeds $s = (E, F)$ becomes

$$\begin{aligned} E' &= \{pre(a) \mid (a, \delta_{adv}) \in F \cup F_{\text{new}}\} \\ F' &= \{(a, \delta - \delta_{adv}) \mid (a, \delta) \in F \cup F_{\text{new}}, \delta > \delta_{adv}\} \end{aligned}$$

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

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

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.

4.2 Properties

The branching rule is sound in the sense that it generates only valid plans, but is not complete; namely, it does not generate *all* valid plans. For example, plans in which no regular action is active during a given time interval are not generated. This is actually a desirable feature.[2] The rule is *optimality preserving* in the sense that it generates *some* optimal plans. This, along with soundness, is all that is needed for optimality (provided an admissible search algorithm and an admissible heuristic are used).

4.3 Heuristic

Like in [7], we derive an admissible planning heuristic by introducing approximations in the recursive formulation of the optimal cost function. 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 Bellman equation [1]:

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

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 $F = (a_i, \delta_i)_i$, at time t must achieve the preconditions of the actions a_i at times $t - \delta_i$ and these preconditions must remain true until t, we must have

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

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

In addition, since achieving a set of atoms E implies achieving each subset E' in 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.

[2] The only plans generated are those in which no-op's start at times in which 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 integer time line and setting δ_{adv} to 1, but this is inefficient and, as mentioned above, not needed for optimality.

Temporal Heuristics H_T^m. As in [7], we define a lower bound H_T^m on the optimal function H^* by transforming the above inequalities into suitable equalities. The family of admissible temporal heuristics H_T^m for an arbitrary $m = 1, 2, \ldots$ is then defined by the equations

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

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

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

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

The relaxation is a result of the last two equations; the first two are also satisfied by the optimal cost function. By unfolding the right-hand side of equation (6) using (8), we obtain that 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, we can solve and precompute $H_T^m(E, \emptyset)$ for all sets of atoms, $|E| \leq m$, and use equations (7) and (8) at run time to compute the heuristic value of arbitrary states.All these operations are tractable; the precomputation is a simple variation of a shortest-path problem and its complexity is a low polynomial in the total 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 be of the form $s = (\{p\}, \emptyset)$, and the regression set $R(s)$ will contain only the states $s' = (pre(a), \emptyset)$ for actions a such that $p \in add(a)$. As a result, for $m = 1$, (6) becomes

$$H_T^1(\{p\}) = \min_{a: p \in add(a)} [dur(a) + H_T^1(pre(a))] \tag{9}$$

where $H_T^1(E)$ stands for $H_T^1(E, \emptyset)$. The planner uses the heuristic H_T^2. The corresponding equations are in [8].

4.4 Search Algorithm

Any admissible search algorithm, *e.g.* A*, IDA* or branch-and-bound [12], can be used with the search scheme described above to find optimal solutions. We have used IDA* with some standard enhancements: a transposition table [16], cycle checking, and an optimality preserving pruning rule explained below. For heuristic we use H_T^2, precomputed for sets of at most two atoms as described above.

4.5 Refinements

The temporal planner we have developed incorporates a number of refinements to the ideas presented above.

Incremental Branching The branching scheme implemented in the planner does not select the set of establishers SE all at once. Instead, this set is constructed incrementally, one action at a time. After each action is added into this set, the cost of the resulting "partial" state is estimated so that dead-ends are detected early (namely, states whose cost exceeds the bound). A similar idea is used in GRAPHPLAN. In the temporal setting, things are a bit more subtle because no-op's have a duration that is not fixed a priori. Still, a lower bound on these durations can be obtained from the durations of the regular actions that participate in the set SE.

Selecting the Atom to Regress The order in which atoms are selected for regression makes no difference for completeness but makes a difference in the size of the resulting search trees. In the planner, we regress atoms p in decreasing order of "difficulty" given by the heuristic estimate $H^2_T(\{p\})$.

Pruning Rule As mentioned above, it is not necessary to generate all valid plans in order to guarantee optimality. The planner refines the branching rule above so that only right-shifted plans are generated. A right-shifted plan is one in which the actions are scheduled as late as possible without affecting the makespan or structure of the plan. Non right-shifted plans are excluded by the following pruning 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-op's. The reason for this is that the action a could have been used to support the same atoms in E, so it could have been shifted to the right (delayed).

5 Planning with Resources

We show next how the planning algorithm deals with renewable and consumable resources.

5.1 Renewable Resources

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

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

for every renewable resource $r \in R_P$.

Heuristic The H_T^m heuristics are admissible in the presence of renewable resources, however, in order to get better lower bounds, we exclude from the regression set any set of actions that violate the corresponding resource constraints. For unary resources (capacity 1) the resulting heuristic H_T^2 remains informative, but for multi-capacity resources the resulting bounds tend to be weak.

5.2 Consumable Resources

For consumable resources, we represent not only the initial amount available $avail(r)$ but also the amount of the resource that is left $rem(s, r)$. For the initial state, $rem(s_0, r) = avail(r)$, while for the state s' that results from s is, for each $r \in C_P$

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

Heuristic The heuristics H_T^m remain admissible in the presence of consumable constraints, yet they become less useful as they predict completion time but not conflicts due to overconsumed resources. However, if we restrict consumable resources to be *monotonically decreasing* (i.e., consumed but not produced), we can estimate the amount of resource r needed to achieve the state s from the initial situation, $need(s, r)$, and prune a state s when $need(s, r) > rem(s, r)$. The amount of resource needed can be estimated with a function $need^m(s, r)$ defined in a way analogous to the heuristic function $H_T^m(s)$ that estimates time. The planner implements the resource estimator $need^m(s, r)$ for $m = 1$.

Because resource consumption is treated separately from time, this solution is not suitable when the time and resources needed to complete a goal interact in complex ways. The H_T^m 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 (for some related ideas see [15]). At the same time, these estimates could be used to optimize some combination of time and resources as opposed to time alone.

5.3 Maintenance Actions

In planning, it's 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. For example, in the spacecraft domain, fuel is consumed to keep the spacecraft pointing in a fixed direction. We'll refer to no-op's that consume resources as *maintenance actions.* We have incorporated maintenance actions in our planner as the branching scheme makes it very simple to do so. For each atom p and each consumable resource r, there is a quantity $use(\texttt{maintain}(p), r)$ which can be provided as part of the domain definition and which is set to 0 by default. Since the duration of no-op's is variable, we assume $use(\texttt{maintain}(p), r)$ to represent a *rate* of consumption. For the rest, maintenance actions are treated as regular actions. No other changes are needed in the planning algorithm.[3]

6 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]. As mentioned above, we use IDA* with some standard enhancements and the H_T^2 heuristic. The resource consumption estimators consider only single atoms.

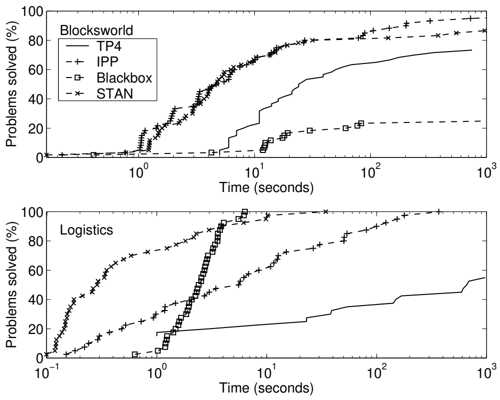
6.1 Non-Temporal Planning

First, we compare TP4 to the 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 4 – 5 deliveries. Blocksworld problems were generated using Slaney & Thiebaux's BWSTATES program [17].

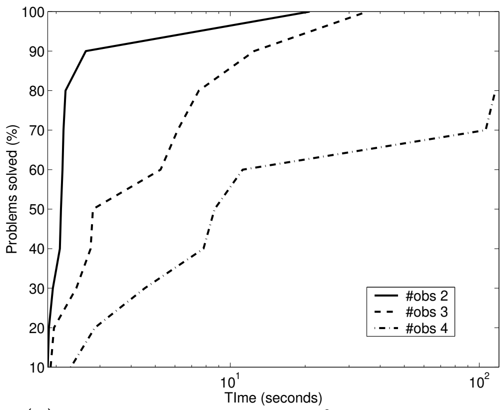
Figure 1(a) presents the results in the form of runtime distributions (a point $\langle x, y \rangle$ on the curve indicates that y percent of the problems were solved

[3] This treatment of maintenance actions is not completely general. Recall that the branching rule does not generate all valid plans, but only plans in which no-op actions start at the same time as some regular action. 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 generated. When this happens, the planning algorithm may produce non-optimal plans or may even fail to find a plan when one exists. This is a subtle issue that we plan to address elsewhere.

[4] TP4 is implemented in C. Planner, problems and 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 three optimal parallel planners on non-temporal problems. Note that the time axis is logarithmic.



(b) Runtime distributions for TP4 in the Spacecraft domain.

Figure1.

in x seconds or less). Clearly TP4 is not competitive with parallel planners. This 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).

6.2 Temporal Planning

We have also tested TP4 on temporal planning problems: Figure 1(b) shows runtime distributions for TP4 on problems from the spacecraft domain with varying a number of observation goals[5]. The average solution complexities are as follows:

# Goals	Makespan	#Actions
2	25.9	7.9
3	31.5	10.9
4	36.3	12.6

Though it is a simplified version of the real problem, this is a very complex domain involving time, renewable and consumable resources and maintenance actions. Plan makespan depends on all choices made in complicated ways

[5] All problems have 15 directions, on average 5 combinations of instrument/calibration target for each observation, and allow on average 1.66 concurrently active instruments. A PDDL description of the domain is in the appendix.

(*e.g.* the sequencing of observations and the choice of instrument both affect the number of calibrations needed, depending on power constraints; the time to calibrate depends on the available reference targets for the chosen instrument). This makes optimal planning hard. We are not aware of any other domain-independent planner that can both model such problems and solve them with reasonable efficiency. The results should be seen as a point of reference for future comparative studies.

We are trying to compare TP4 and TGP [19] in temporal domains, but have had difficulties running TGP and have no results yet.

7 Conclusions

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

The planner achieves a tradeoff between the performance of modern STRIPS planners and the expressivity of planners such as IxTeT and HSTS. Yet it is not competitive with either the best parallel planners or specialized schedulers. An approach 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 [5] for details.

Acknowledgments

This research has been supported by ??????, by the Wallenberg Foundation, Sweden, and the ECSEL/ENSYM graduate study program.

A Appendix: Autonomous Spacecraft Domain

The following are the main points in our formulation of the domain described in [18] and used in the experiments.

```
(define (domain spacecraft0)
 (:requirements :strips :typing)
 ...
 (:action turn_to
   :parameters (?d_new - direction)
   :vars (?d_prev - direction)
   :precondition (and (pointing ?d_prev) (not (= ?d_prev ?d_new)))
   :effect (and (not (pointing ?d_prev)) (pointing ?d_new))
   :resources ((fuel 3.0))
   :duration ?t : (time_to_turn ?d_prev ?d_new ?t))

 (:action maintain_direction
   :parameters (?d - direction)
   :precondition (pointing ?d)
   :effect (pointing ?d)
   :resources ((fuel 0.5)))

 (:action switch_on
  :parameters (?i - instrument)
  :effect (and (power_on ?i) (not (calibrated ?i)))
  :resources ((power ?p : (power_required ?i ?p)))
  :duration 2.0)

 (:action maintain_power
  :parameters (?i - instrument)
  :precondition (power_on ?i)
  :effect (power_on ?i)
  :resources ((power ?p : (power_required ?i ?p))))

 (:action calibrate
  :parameters (?i - instrument ?d - direction)
  :precondition (and (power_on ?i) (pointing ?d)
                     (calibration_target ?i ?d))
  :effect (calibrated ?i)
  :duration 1.0)

 (:action take_image
  :parameters (?i - instrument ?q - quality ?d - direction)
  :precondition (and (power_on ?i) (calibrated ?i) (capability ?i ?q)
                     (pointing ?d))
  :effect (have_image ?d ?q)
  :duration 1.0))
```

The complete domain, along with a random problem generator, is available from `http://www.ida.liu.se/~pahas/hsps/`.

References

1. R.E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.
2. A.L. Blum and M.L. Furst. Fast planning through graph analysis. *Artificial Intelligence*, 90(1–2):281 – 300, 1997.
3. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001. To appear.
4. 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.
5. H. Geffner. Planning as branch and bound and its relation to constraint-based approches. `http://www.ldc.usb.ve/~hector`.
6. M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proc. 2nd International Conference on AI Planning Systems*, 1994.
7. P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling.* AAAI Press, 2000.
8. P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proc. IJCAI Workshop on Planning with Resources*, 2001. To appear. `http://www.ida.liu.se/~pahas/hsps/`.
9. 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.
10. 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.
11. J. Koehler. Planning under resource constraints. In *Proc. 13th European Conference on Artificial Intelligence*, 1998.
12. R.E. Korf. Artificial intelligence search algorithms. In *Handbook of Algorithms and Theory of Computation*, chapter 36. CRC Press, 1999.
13. N. Muscettola. Integrating planning and scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling.* Morgan-Kaufmann, 1994.
14. J.S. Penberthy and D.S. Weld. Temporal planning with continous change. In *Proc. 12th National Conference on Artificial Intelligence (AAAI'94)*, 1994.
15. I. Refanidis and I. Vlahavas. Heuristic planning with resources. In *Proc. 14th European Conference on Artificial Intelligence*, 2000.
16. A. Reinfeld and T. Marsland. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7):701 – 710, 1994.
17. J. Slaney and S. Theibaux. Blocks world revisited. *Artificial Intelligence*, 125, 2001. See `http://arp.anu.edu.au:80/~jks/bw.html`.
18. D.E. Smith, J. Frank, and A.K. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.
19. D.E. Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. 16th International Joint Conference on Artificial Intelligence*, 1999.
20. 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.