Planning under Resource Constraints

Jana Koehler[1]

Abstract. This paper outlines the basic principles underlying reasoning about resources in IPP, which is a classical planner based on planning graphs originally introduced with the GRAPHPLAN system. The main idea is to deal with resources in a strictly action-centered way, i.e., one specifies how each action consumes or produces resources, but no explicit temporal model is used. This avoids the computational problems of solving general constraint satisfaction problems by using instead interval arithmetics and propagation of resource requirements over time steps in the planning graph.

1 Actions that provide, produce, and consume Resources

The starting point for the language extension is the ADL subset that is available in IPP 3.0 [7]. It offers universally quantified and conditional effects, atomic negation, equality as well as quantified and conditional goals. To reason about resources, an action description is extended in the following way:

1. Following the "ordinary preconditions" (which are logical facts) *resource requirements* can be specified.
2. Effect descriptions are extended to specify which of the resource variables is *provided*, *produced* or *consumed* by the action.
3. Database query schemata allow for a compact representation of resource requirements and resource effects.

As an example, consider the following two actions that are part of our version of the *airplane* example used by the ZENO planner [9].

```
fly(?x ?y:airport)
:pre plane-at(?x) ($gas ≥ distance(?x ?y)/3)
:eff plane-at(?y) not plane-at(?x);
     ALL ?p:passenger boarded(?p)
             => at(?p ?y) not at(?p ?x);
     $gas  -= distance(?x ?y)/3;
     $time += distance(?x ?y) * (3/20).

refuel(?x:location)
:pre plane-at(?x)
:eff $gas  := 750;
     $time += -0.08*$gas+60 | 60.
```

The action **fly** specifies what happens when an airplane flies from airport x to airport y. As a precondition it needs to be at airport x and as a resource requirement there must be enough gas in the tank to fly the distance between the two airports specified with `$gas` $\geq$ `distance(?x ?y) / 3`. We allow the distinguished binary predicates $=, \leq, \geq$, which mean that the current amount of the resource variable `$gas` specified as the first argument must be either *equal*, *lower-or-equal*, or *higher-or-equal* the arithmetic expression following as the second argument. An arithmetic expression can contain a database query schema such as `distance(?x ?y)` together with an arithmetic term built out of the four basic mathematical operations and real numbers. When the action gets instantiated, the query schema is instantiated into a database query such as `distance(Basel Paris)` for which the specific value is found in the associated database. Together with the arithmetic term it can be simplified to a single real number yielding `$gas` $\geq$ 200 as the resource requirement of the action instance **fly(Basel Paris)**.[2]

As an effect of the action, `$gas` is consumed (indicated by the assignment operator `-=`) and `$time` is "produced" (indicated by `+=`), i.e., it advances on a discrete scale of time units when actions are executed. Producers (`+=`) and consumers (`-=`) increase or decrease a resource relative to its current value, e.g., if the action **fly(Basel Paris)** consumes 200 units of gas and the current value of this resource is 300 then the new value of `$gas` results with 100.

Two more kinds of effects are allowed and shown in the action **refuel**. The first kind of effect assigns to `$gas` the absolute value of 750 independently of its previous value (indicated with the assignment operator `:=`), i.e., **refuel** is a *provider* of `$gas`. The second kind of effect specifies that the time for refuelling depends on the gas amount that is needed by the airplane. We allow for a limited form of interdependencies between resource variables where one variable can depend on at most another one via a linear equation of the form $y = mx + b$. Effects without resource interdependencies are referred to as *simple*. The action also shows an expression of the form |NUMBER, which represents a *worst-case value*, i.e., `$time` will at most be increased by 60 minutes (if this is the assumed unit) when refilling the maximum amount of 750. Currently, we do not allow interdependency chains where `$x` depends on `$y`, which again could depend on `$z` etc.

2 Planning Problems involving Resources

The specification of a planning problem includes the usual declaration of constants and their types, i.e.,

[1] Albert Ludwigs University, Am Flughafen 17, 79110 Freiburg, Germany, koehler@informatik.uni-freiburg.de

[2] For each resource variable a unit of measurement is assumed that remains implicit in the planner.

© 1998 J. Koehler
ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

```
passenger: Dan Ernie Scott;
location:  Paris Basel London;
```

a declaration $[dmin(\$r), dmax(\$r)]$ of the minimum and maximum values each resource variable can take

```
$time: [0, ∞);
$gas:  [0,750];
```

and the database information (since no connection to a real database is implemented yet) such as

```
database: distance(Paris Basel)    600
          distance(Paris London)   400
          distance(Basel Paris)    600
          distance(Basel London)   800.
```

The specification of the initial state specifies the usual logical facts and assigns values to *all* resource variables that are involved in the planning problem

```
initial: $time=0 $gas=300
         checked-in(Ernie) checked-in(Scott)
         plane-at(Basel) at(Ernie Paris)
         at(Dan Paris) at(Scott Basel).
```

The goal specification contains resource requirements together with logical facts that have to hold in the goal state:

```
goal: $time ≤ 330
      at(Ernie London) at(Scott London).
```

This states the planning problem to fly two passengers from two different locations to London in less than 330 minutes.

3 Resource Time Maps

To build planning graphs, all valid instances of applicable actions are determined. The instantiation of a database query schema is a specific query returning the value from the database. This means, an instantiated action can only require a specific numerical value as a resource requirement in its preconditions such as $\$time \leq 100$ or $\$gas = 50$. Resource effects are reduced to an operator followed by a numerical value or to an operator followed by a linear equation containing one resource variable.

For each instantiated action a, its specific requirements for resource $\$r$ as the interval $[rmin(a, \$r), rmax(a, \$r)]$ and a computational instruction for each resource that it affects are determined. If no resource requirement is specified, the action is applicable if the resource takes an arbitrary value specified by $(-\infty, +\infty)$. For the example, we obtain

action		$gas	$time
fly(B P)	pre:	$[200, +\infty)$	$(-\infty, +\infty)$
	eff:	$-= 200$	$+= 90$
fly(B L)	pre:	$[266.67, +\infty)$	$(-\infty, +\infty)$
	eff:	$-= 266.67$	$+= 120$
fly(P L)	pre:	$[133.33, +\infty)$	$(-\infty, +\infty)$
	eff:	$-= 133.33$	$+= 60$
refuel(B)	pre:	$(-\infty, +\infty)$	$(-\infty, +\infty)$
	eff:	$:= 750$	$+= -0.08 * \$gas$ $+60 \mid 60$
board(P)	pre:	$(-\infty, +\infty)$	$(-\infty, +\infty)$
	eff:		$+= 30$

Providers can assign arbitrary values to resources, while producers and consumers are limited to positive values. For example, one cannot declare `$gas += -500`, i.e., that an action is a producer, but the instruction would in fact decrement the value of the resource. Linear equations are allowed for producers and consumers only and also have to return positive values.

The first layer in the planning graph is obtained from the logical facts in the initial state specification. Together with the graph, a *resource time map* RTM is built that records the possible interval boundaries $[tmin(n, \$r), tmax(n, \$r)]$ for each resource variable $\$r$ at each time step n and that is similar to the resource utilisation manager in O-Plan [3]. The initialisation of the intervals at time step 0 is based on the specification of the initial state. For example `$money` $\geq$ `200` leads to $[200, +\infty)$ and `$debts` $\leq$ `10, $debts` $\geq$ `5` lead to $[5, 10]$. This way, one can also represent uncertainty about the exact amount of a resource that is initially available.

Definition 1 *Let $[rmin(a, \$r), rmax(a, \$r)]$ be the resource requirement of action a and $[tmin(n, \$r), tmax(n, \$r)]$ be the RTM interval for $\$r$ in time step n, the resource requirement is* possibly satisfiable *in n iff $[rmin(a, \$r), rmax(a, \$r)] \cap [tmin(n, \$r), tmax(n, \$r)] \neq \emptyset$.*

Definition 2 *A simple resource effect $\$r$ OP N is* impossible *iff*

$$\begin{array}{ll} tmin(n, \$r) + \mathsf{N} > dmax & \textit{if } \mathsf{OP} \textit{ is } += \\ tmax(n, \$r) - \mathsf{N} < dmin & \textit{if } \mathsf{OP} \textit{ is } -= \\ \mathsf{N} \notin [dmax(\$r), dmin(\$r)] & \textit{if } \mathsf{OP} \textit{ is } := \end{array}$$

Definition 3 *An action a is* applicable *to a fact level n in the planning graph iff*
1. its logical preconditions are non-exclusive *in n*
2. all its resource requirements are possibly satisfiable *in n*
3. none of its simple resource effects is impossible.

The exclusivity relation over pairs of actions is extended in such a way that a parallel set of actions causes the same resource effects independently of a particular linearization of the actions. This restriction eliminates all possible resource conflicts within parallel actions and leads to a unique resulting state with respect to resources independently of the execution order of the actions.

Definition 4 *Two actions are marked as* exclusive *if one of the following holds*
- *they are logically exclusive [7]*
- *they belong to different action types (consumers, providers, and producers are exclusive of each other)*
- *both actions are providers of the same resource*
- *one affects a resource variable that occurs in the linear equation of the other's effects*
- *their combined simple effects are impossible (Def. 2)*
- *both have contradictory resource requirements (the intersection of the corresponding resource intervals is empty).*

The next fact level in the graph is built as usual, i.e., all effects whose effect conditions can possibly be made true are added to the level and the appropriate ADD and DEL edges are drawn. Logical facts are marked as exclusive as before, see [7]. Resource effects update the resource time map at the next level based on the following rules:

Definition 5 *Given the interval* $[tmin(n,\$r), tmax(n,\$r)]$ *of resource \$r at time step n in the RTM, to compute the new interval* $[tmin(n+1,\$r), tmax(n+1,\$r)]$ *at time step* $n+1$ *we do the following:*
(1) If no applicable action affects \$r then
$[tmin(n{+}1,\$r), tmax(n{+}1,\$r)] = [tmin(n,\$r), tmax(n,\$r)]$.
(2) Otherwise, for all actions a that affect \$r: If the effect of a on \$r contains a linear equation, it is replaced by the worst-case value, i.e., each resource effect is simplified to $\mathsf{OP}\ \mathsf{N}_{(\mathsf{a})}$. *For the distinguished resource* `$time`*, which is only "produced"*

$$\begin{array}{rcl} tmin(n+1,\$time) & = & tmin(n,\$time) \\ tmax(n+1,\$time) & = & \underset{a}{\mathbf{MAX}}\ [tmax(n,\$time) + \mathsf{N}_{(\mathsf{a})}] \end{array}$$

For all other resource variables \$r the algorithm computes the interval boundaries for maximal sets of non-exclusive actions that affect a resource, i.e., given an action it adds to it all actions that are non-exclusive following Def. 4. This set construction is repeated for each action that occurs at the current level in the graph.
Each set s of k non-exclusive producers *of \$r yields one new upper interval boundary*

$$tmax(n+1,\$r)_s = tmax(n,\$r) + \textstyle\sum_{a=1}^{k} \mathsf{N}_{(\mathsf{a})}$$

Each set s of k non-exclusive consumers *of \$r yields one new lower interval boundary*

$$tmin(n+1,\$r)_s = tmin(n,\$r) - \textstyle\sum_{a=1}^{k} \mathsf{N}_{(\mathsf{a})}$$

Each single provider *a of \$r yields another minimum and maximum interval boundary*

$$tmin(n+1,\$r)_s = tmax(n+1,\$r)_s = \mathsf{N}_{(\mathsf{a})}$$

Furthermore, the interval boundaries at time step n are propagated to time step $n+1$ *to yield another pair of interval boundaries:*

$$tmin(n+1,\$r)_s = tmin(n,\$r)$$
$$tmax(n+1,\$r)_s = tmax(n,\$r)$$

The new RTM interval for \$r at time step $n+1$ *results as the minimum and maximum values from the set of new interval boundaries intersected with the predeclared interval.*

$$[tmin(n+1,\$r), tmax(n+1,\$r)] = [dmin(\$r), dmax(\$r)] \\ \cap [\underset{s}{\mathbf{MIN}}\ tmin(n+1,\$r)_s, \underset{s}{\mathbf{MAX}}\ tmax(n+1,\$r)_s]$$

Let us consider the following example: In the initial state (fact level 0) the RTM is initialised with [0,0] for `$time` and [300,300] for `$gas`. The initially applicable actions are: **fly(Basel Paris)**, **fly(Basel London)**, **refuel(Basel)**, **board(Basel)**. For `$time` we only have producers which add 30, 60, 90 or 120 minutes each, i.e., at fact level 1 we end up with the interval [0,120] taking the maximum increase. The amount of gas is set to 750 by refuelling, which is a provider, but flying actions are consumers decreasing the variable by either 266.67, 200 or 133.33. The *flying* actions are marked as exclusive for two reasons: first, they interfere with respect to their logical effects and second, their combined simple resource effects fall below the valid interval minimum of 0 for `$gas`. Therefore, five individual new lower bounds are obtained—the old value 300, one from each flying action, and one from the provider—of which the minimum is selected: $\mathbf{MIN}(300, 300-200, 300-133.33, 300-266.67, 750) = 33.33$. The new upper bound results from the maximum of the old value with 300 and the assignment of the provider with 750. Thus $[tmin(1,\$gas), tmax(1,\$gas)]$ is the intersection of $[0,750] \cap [33.33,750] = [33.33,750]$.

This way, the graph is expanded until the logical goals are reached for the first time without being exclusive and the RTM intervals reach a non-empty intersection with all goal intervals.

4 Finding a Valid Choice of Resource-constrained Parallel Actions

Recall that only a set of non-exclusive actions can be selected at each time step when we are searching from the goal level back to the initial state. For resources, non-exclusivity implies that at each time step a resource can either be consumed, produced, or provided and that never a dependency resource can be changed simultaneously with the depending resource.

The planning algorithm comprises two parts: The *candidate generation*, which searches the planning graph and RTM for a plan that *possibly* solves the planning problem and the *symbolic execution* that proves that the generated candidate is indeed correct.

The generation algorithm searches the planning graph level by level starting from the goals. It works in two phases: In a first action selection phase, it chooses a minimal non-exclusive and conflict-free set of parallel actions to achieve the logical goals. If the resulting new resource goals are inconsistent with the individual resource requirements of a selected action or the maximal possible range of resources as reflected in the RTM, a conflict has occurred. In this case, a second phase adds additional actions to the minimal set in order to achieve a conflict resolution. In the following, we solely concentrate on the resource-side of each action selection, how the process proceeds for the logical goals is described in [7].

At each time step $n+1$, the algorithm is given the logical goals G_{n+1} and a resource goal $[gmin(n+1,\$r), gmax(n+1,\$r)]$ for each resource variable \$r that occurs in the planning problem. It is initially called starting at the *max* time step of the graph with the logical goals that have been specified in the planning problem. If any resource goals have been specified too, they are intersected with the corresponding RTM interval to yield the resource goals at time step *max*. If no specific value for a resource is required in the goal state, then the resource goal is initialised with the RTM interval at time step *max*. Action selection proceeds as follows:

(1) We start with an initialisation: the set of selected actions Δ_n is initially empty and all new resource goals at time step n are initialised with the resource values from time step $n+1$.

$$\Delta_n = \emptyset$$
$$\forall\ \$r : [gmin(n,\$r), gmax(n,\$r)] := \\ [gmin(n+1,\$r), gmax(n+1,\$r)]$$

(2) An action a is selected that achieves a goal from G_{n+1} and that is non-exclusive to the actions already contained in Δ_n and $\Delta_n = \Delta_n \cup \{a\}$. If no selection is possible the search algorithm backtracks to action level $n+1$.

(3) For each resource \$r, the following tests are performed with the selected action a and the resource effects of a to update the resource goals:

```
if $a$ does not affect $r: do nothing
elsif $a$ is provider of $r
then if $gmin(n+1,\$r) \le \mathsf{N} \le gmax(n+1,\$r)$
```

```
        then $gmin(n, \$r) := -\infty$
                $gmax(n, \$r) := +\infty$
        else backtrack to select a new action
else
$a$ is simple-effect producer/consumer and $\$r \neq \$time$:
$gmin(n, \$r) := gmin(n, \$r)\ \overline{\mathsf{OP}}\ \mathsf{N}_{(a)}$
$gmax(n, \$r) := gmax(n, \$r)\ \overline{\mathsf{OP}}\ \mathsf{N}_{(a)}$
$a$ is simple-effect producer and $\$r = \$time$:
$gmin(n, \$r) := gmin(n+1, \$r) - \mathbf{MAX}_{a \in \Delta_n} \mathsf{N}_{(a)}$
$gmax(n, \$r) := gmax(n+1, \$r) - \mathbf{MAX}_{a \in \Delta_n} \mathsf{N}_{(a)}$
$a$ affects $r with non-simple effect: do nothing
(the exact value returned by the equation is yet unknown)
```

with $\overline{\mathsf{OP}}$ being the inverse operator to OP, i.e., we use $+$ instead of $-$ and vice versa. $\mathsf{N}_{(a)}$ is the numerical value from the simple effect. For example, if the old goal was $[100, +\infty)$ and the effect is `-= 50` then we obtain $[100 + 50, +\infty + 50)$ which yields the new goal interval $[150, +\infty)$.

This way, actions are selected until a minimal action set is found that achieves all logical goals. The updated resource goals for the current choice of actions are tested against the resource requirements of each action and the RTM intervals at time step n:

Test (1): $\exists\ \$r, a \in \Delta_n : [gmin(n, \$r), gmax(n, \$r)] \cap [rmin(a, \$r), rmax(a, \$r)] = \emptyset$

Test (2): $\exists\ \$r : [gmin(n, \$r), gmax(n, \$r)] \cap [tmin(n, \$r), tmax(n, \$r)] = \emptyset$

If the tests fail, i.e., all intersections are non-empty, the final new resource goals for this minimal action set are obtained from the intersection of the goals with all resource requirement intervals of all actions in Δ_n and the RTM interval:

$$[gmin(n, \$r), gmax(n, \$r)] := [gmin(n, \$r), gmax(n, \$r)] \cap \bigcap_{a \in \Delta_n} [rmin(a, \$r), rmax(a, \$r)] \cap [tmin(n, \$r), tmax(n, \$r)]$$

Together with the new logical goals resulting from the preconditions and effect conditions of selected actions, these resource goals are forwarded to the next level of the planning graph. The algorithm terminates successfully in the initial state if $[tmin(0, \$r), tmax(0, \$r)] \subseteq [gmin(0, \$r), gmax(0, \$r)]$ for all resources $\$r$.

If one of the above tests succeeds, a resource conflict occurs that the planner needs to resolve by adding more actions such that the new goals of the final action set will fall within predeclared interval boundaries and the recorded RTM intervals, and are consistent with the resource requirements of each action. Depending on why a test was successful, a conflict resolution policy is determined:
(A) If $gmax(n, \$r) < T$ where T stands for the interval against which the goals are tested, then a consumer or provider of $r is added to Δ_n in the *next* selection.
(B) If $gmin(n, \$r) > T$ then a producer or provider of $r is added to Δ_n in the *next* selection.

This means that it is no longer sufficient to restrict the search space of the planner to action sets that are minimal wrt. the logical goals. It is easy to see that there can be actions that must solely be selected to achieve resource goals, but are not necessary at all to achieve the logical goals. To achieve completeness, all possible minimal action sets that achieve a set of logical goals are generated and for each of the sets all possibilities to add conflict resolvers have to be explored. The algorithm returns the first conflict-free set of actions that it finds, which does not need to be the smallest set in terms of the number of actions. Furthermore, if a conflict cannot be resolved at time step n then the planner has to explore if it can prevent the conflict at levels $i \geq n$.

The necessity to explore all possibilities for conflict resolution by addition of actions at various levels in the graph leads to a large search space that is only limited by the rather strict exclusivity relations that we formulated over actions, the number of actions at each time step in the planning graph, and the depth of the graph - more precisely the number of action levels between max and the level n where the conflict was detected. If the complete search over a graph of depth t has failed in generating valid plans, the planning graph and RTM are extended to depth $t + 1$ and the planner searches again the extended graph.

If no linear equations involving resource dependencies occur in the chosen actions then this algorithm is a sound and complete planner. Otherwise, the symbolic execution phase needs to verify that a generated plan is indeed a solution.[3]

5 An Example

In the airplane example, the planner starts with the goal intervals for `$time` $[0, 330]$ (resulting from the intersection of the original goal $(-\infty, 330]$ and the RTM interval) and `$gas` $[0, 750]$ (resulting from the RTM interval because no value is specified) at fact level 4 (the planner has unsuccessfully tried to search a plan starting from level 3, i.e., the planning graph and RTM were extended by one more level). To achieve the goals `at(Ernie London)` and `at(Scott London)` the action **fly(Paris London)** is selected, which leads to the following goals at fact level 3:

```
f 3: $time ∈ [0,270] $gas ∈ [133.33,750]
     boarded(Ernie) boarded(Scott) plane-at(Paris)
```

For `$time` the planner computes $[0 - 60, 330 - 60]$ intersected with the resource requirements $(-\infty, +\infty)$ and the RTM $[0, 360]$. For `$gas` it computes $[0 + 133.33, 750 + 133.33]$ intersected with the resource requirements $[133.33, +\infty)$ and the RTM interval $[0, 750]$.

Now, the planner selects **board(Paris)** for Ernie to enter the aircraft, while `boarded(Scott)` and `plane-at(Paris)` are forwarded to time step 2. Boarding takes 30 minutes and consumes no gas, i.e., the new time goal is $[0 - 30, 270 - 30]$ intersected with $(-\infty, +\infty)$ and $[0, 240]$.

```
f 2: $time ∈ [0,240] $gas ∈ [133.33,750]
     checked-in(Ernie) at(Ernie Paris)
     plane-at(Paris) boarded(Scott)
```

To achieve the goal `plane-at(Paris)`, the action **fly(Basel Paris)** is selected, while all other subgoals are forwarded to time step 1. Flying requires 200 amounts of gas and takes 90 minutes, i.e., the planner gets for `$time` the intersection of $[0-90, 240-90]$ with $(-\infty, +\infty)$ and $[0, 120]$. For `$gas` it intersects $[133.33 + 200, 750 + 200]$ with $[200, +\infty)$ and $[33.33, 750]$ which yields $[333.33, 750]$.

[3] A more detailed description of the planning algorithm and the implemented prototype can be obtained from the IPP homepage: http://www.informatik.uni-freiburg.de/˜ koehler/ipp.html.

```
f 1: $time ∈ [0,120] $gas ∈ [333.33,750]
     checked-in(Ernie) at(Ernie Paris)
     plane-at(Basel) boarded(Scott)
```

Finally, Scott needs to board in Basel before the airplane takes off for Paris, i.e., the planner selects **board(Basel)** as the action that will be executed in the initial state. For `$time` the new goal interval in the initial state is $[0-30, 120-30]$ intersected with $(-\infty, +\infty)$ and $[0, 0]$, which yields $[0, 0]$. For `$gas` the planner intersects $[333.33, 750]$ with $[300, 300]$, which is empty. This alerts the planner that a resource conflict has occurred on `$gas`. Since the goal interval lies "right" of the RTM interval, a producer or provider has to be considered for conflict resolution. The refuelling action is the only provider of `$gas` and fortunately, it is non-exclusive of boarding and satisfies the test $333.33 \leq 750 \leq 750$. The new goal for this resource is recomputed as $[-\infty, +\infty]$, which is intersected with $[300, 300]$ to yield the resource goal in the initial state. Obviously, this goal interval coincides perfectly with the amount of gas that is available in the initial state.

```
f 0: $time ∈ [0,0] $gas ∈ [300,300]
     checked-in(Ernie) at(Ernie Paris)
     plane-at(Basel) checked-in(Scott)
```

Since a non-simple resource effect occurs—the time for refuelling depends on the amount of gas that is available in the initial state—this plan just represents a candidate and has to be verified by symbolic execution, i.e., the planner computes how each action exactly affects the resources in each time step. If (1) all resource requirements of all actions are satisfied in the state in which they are scheduled for execution, (2) all their effects yield resource values that fall within the predeclared intervals, and (3) if the resource goals hold in the final state, then the plan is indeed a solution.

6 Empirical Evaluation

In contrast to classical STRIPS planning, only very few approaches exist that try to address the problem of resource-constrained planning and their underlying representation formalisms vary to a high degree. Thus, no standardised test suite is available to empirically evaluate this algorithm and compare it to other implementations. Currently, we have some very preliminary results for the first prototypical implementation running on a test suite that was compiled from examples from the literature as well as self-developed test problems.

Some speed-up can be observed compared to other results as reported in the literature. For example, the airplane problem as described in this paper is solved by IPP in 0.03 seconds (without any user interaction) compared to 2-3 minutes for ZENO [9] with user-guided goal selection.

For the missionaries and cannibals problem, the optimal plan of 21 actions is found in approx. 2 minutes and an example with a blocks-manipulating and fuel-consuming robot taken from [11] that requires to generate a plan of 4 actions is solved in 0.04 s.

7 Conclusion

This paper describes the first attempt of dealing with resources in IPP. The approach is strictly action-centered and follows the classical planning paradigm, i.e., a logical goal is achieved by constructing a valid plan, but resources are no longer unlimited. In this formalism, we cannot specify that a resource has a certain value at a specific time point because no explicit temporal model is available as for example in [1, 11, 9, 4, 5, 8, 3, 2]. We decided to adopt this limitation in order to avoid the necessity to solve general constraint satisfaction problems during planning since the available algorithms are excellent satisfiability checkers, but usually fail in their generative capabilities, i.e., in actually constructing solution plans. First empirical tests show that this limitation in expressivity directly translates into efficiency gains and allows to solve interesting planning problems in reasonable time.

We allow for a limited form of temporal reasoning by treating time as a distinguished resource. Actions produce time, but they are still treated as instantaneous in the planning graphs. It has to be noted that this can be adequate only for some applications. We consider it as a preliminary step towards an extension of planning graphs to model action duration.

REFERENCES

[1] J. Allen, H. Kautz, R. Pelavin, and J. Tenenberg, *Reasoning about Plans*, Morgan Kaufmann, USA, 1991.

[2] A. Cesta and C. Stella, 'A time and resource problem for planning architectures', In Steel [10], pp. 117–129.

[3] B. Drabble and A. Tate, 'The use of optimistic and pessimistic resource profiles to inform search in an activity based planner', In Hammond [6], pp. 243–248.

[4] A. El-Kholy and Barry Richards, 'Temporal and resource reasoning in planning: the parcPLAN approch', in *Proceedings of the 12th European Conference on Artificial Intelligence*, ed., W. Wahlster, pp. 614–618. John Wiley & Sons, Chichester, New York, (1996).

[5] M. Ghallab and H. Laruelle, 'Representation and control in IxTeT, a temporal planner', In Hammond [6], pp. 61–67.

[6] K. Hammond, ed. *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems.* AAAI Press, Menlo Park, 1994.

[7] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos, 'Extending planning graphs to an ADL subset', In Steel [10], pp. 273–285.

[8] P. Laborie and M. Ghallab, 'Planning with sharable resource constraints', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1643–1649. Morgan Kaufmann, San Francisco, CA, (1995).

[9] J. Penberthy and D. Weld, 'Temporal planning with continuous change', in *Proceedings of the 12th National Conference of the American Association for Artificial Intelligence*, pp. 1010–1015. AAAI Press, MIT Press, (1994).

[10] S. Steel, ed. *Proceedings of the 4th European Conference on Planning*, volume 1348 of *LNAI*. Springer, 1997.

[11] D. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, San Francisco, 1988.