Solving Resource-Constrained Planning Tasks

Jana Koehler
Institut für Informatik
Albert Ludwigs Universität
Am Flughafen 17
79110 Freiburg
koehler@informatik.uni-freiburg.de

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[1] 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.

[1] To avoid confusion, we speak of actions instead of planning operators. The term operator is reserved to denote a mathematical operator in an arithmetic expression.

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

refuel(?x:location)
:p at-plane(?x)
:e ADD full-plane();
   $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 `$gas` specified as the first argument must be either *equal*, *lower-than*, or *greater-than* the arithmetic expression following as the second argument. An arithmetic expression can contain a database query schema such as `distance(?x ?y)` together with an arithmetic term built out of the four basic mathematical operations and real numbers. When the action gets instantiated, the query schema is instantiated into a database query such as `distance(Basel Paris)` for which the specific value is found in the associated database. Together with the arithmetic term it can be simplified to a single real number yielding `$gas` $\geq$ `200` as the resource requirement of the action instance **fly(Basel Paris)**.[2]

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

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 `$gas` is decreased by 200 by the action **fly(Basel Paris)** and its current value is 300 then the new value of `$gas` results with 100.

Two more kinds of effects are allowed and shown in the action **refuel**. First, it 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`. Second, the time for refueling 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 atmost another one via a linear equation of the form $y = mx + b$. Effects without resource interdependencies are are referred to as *simple*.

In the first prototype, we do not allow interdependency chains where `$x` depends on `$y`, which again could depend on `$z` etc. The action also shows an expression of the form |NUMBER, which represents a *worst-case value*, i.e., `$time` will at most be increased by 60 minutes (if this is the assumed unit) when refilling the maximum amount of 750.

2 Planning Problems involving Resources

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

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

a declaration [dmin,dmax] 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)
          at-plane(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, i.e., $5\frac{1}{2}$ hours.

3 Planning Graphs with 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.

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. The initialization 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` leads to [5,10]. This way, we can also represent uncertainty about the exact amount of a resource that is initially available.

time step	$gas	$time
0	[300,300]	[0,0]

From each action instance a (in the following denoted with action), its specific requirements for resource $\$r$ (represented as an 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)	p: $[200, +\infty)$	$(-\infty, +\infty)$
	e: $- = 200$	$+ = 90$
fly(B L)	p: $[266.67, +\infty)$	$(-\infty, +\infty)$
	e: $- = 266.67$	$+ = 120$
fly(P L)	p: $[133.33, +\infty)$	$(-\infty, +\infty)$
	e: $- = 133.33$	$+ = 60$
refuel(B)	p: $(-\infty, +\infty)$	$(-\infty, +\infty)$
	e: $:= 750$	$+ = -0.08 * \$gas$ $+60 \mid 60$
board(P)	p: $(-\infty, +\infty)$	$(-\infty, +\infty)$
board(B)	e:	$+ = 30$

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 providers, producers, and consumers, but also have to return positive values for the latter two action types.

Definition 1 *Given a resource requirement* $[rmin(a, \$r), rmax(a, \$r)]$ *of an action* a *wrt. a resource* $\$r$ *and the RTM interval* $[tmin(n, \$r), tmax(n, \$r)]$ *for* $\$r$ *at* n*, the resource requirement is* possibly satisfiable *at* 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*

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

Definition 3 *An action* a *is* applicable *to a fact level* n *in the graph iff*

1. *its logical preconditions are non-exclusive in* n *and*
2. *all its resource requirements are* possibly satisfiable *in* n *and*
3. *none of its simple 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*

1. *they are exclusive wrt. their logical effects or preconditions [7]*
2. *they belong to different action types, i.e., consumers, providers, and producers are exclusive*
3. *both actions are providers of the same resource*
4. *one affects a resource variable that occurs in the linear equation of the other's effects*
5. *their combined simple effects are impossible based on Definition 2.*
6. *both have contradictory resource requirements, i.e., the intersection of the corresponding intervals for the same resource is empty.*

The next fact level in the graph is built as usual, i.e., all effects whose effect conditions can probably 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* $[tmin(n, \$r), tmax(n, \$r)]$ *as the interval for a resoure* $\$r$ *at time step* n *in the RTM, to compute the new interval* $[tmin(n+1, \$r), tmax(n+1, \$r)]$ *for* $\$r$ *at time step* $n+1$ *we do the following:*
(1) If no action a *affects the resource* $\$r$ *at time step* n *then* $[tmin(n+1, \$r), tmax(n+1, \$r)] = [tmin(n, \$r), tmax(n, \$r)]$.
(2) Otherwise, for all actions a *that affect the resource* $\$r$*: 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* $\mathsf{OP}\ \mathsf{N}_{(\mathsf{a})}$.
If $\$r$ *is the distinguished resource variable* `$time`*, which can only be "produced"*

$$tmin(n+1, \$r) = tmin(n, \$r)$$
$$tmax(n+1, \$r) = \mathbf{MAX}_a\ [tmax(n, \$r) + \mathsf{N}_{(\mathsf{a})}]$$

For all other resource variables $\$r$ *we compute the interval boundaries for maximal sets of non-exclusive actions that affect a resource, i.e., given an action the search algorithm adds to it all actions that are non-exclusive and whose combined simple resource effects are possible. This set construction is repeated for all actions that occur at an action level in the graph.*
For each set s *of* k *non-exclusive* producers *of* $\$r$ *a new upper interval boundary*

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

For each set s *of* k *non-exclusive* consumers *of* $\$r$ *a new lower interval boundary*

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

For all providers a *of* $\$r$

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

We obtain individual interval boundaries for each maximal set of non-exclusive producers and consumers and the largest absolute increase or decrease of a resource that can be achieved when a single provider is selected.

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 interval for the resource $\$r$ at time step $n+1$ in the RTM is now obtained as

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

The intersection between an RTM interval and a predeclared interval is never empty if it was non-empty in the initial state because RTM intervals grow monotonically and an empty intersection in the initial state is interpreted as an inconsistent planning problem specification for which no planning takes place.

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

RTM:

fact level	$gas	$time
0	[300,300]	[0,0]
1	[33.33,750]	[0,120]
2	[0,750]	[0,240]
3	[0,750]	[0,360]
4	[0,750]	[0,480]

This way, the graph is expanded until the logical goals are reached for the first time without being exclusive. For the resource requirements we test if the interval from the time map and the goal interval have a non-empty intersection. In the example, both tests are satisfied at fact level 3.

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, this 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 is *possibly* a solution to 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 which leads to resource requirements that are consistent with the individual requirements of each action and the possible values of resources as reflected in the RTM. If a conflict occurs, additional actions are added 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 initialized with the value of the resource as recorded in the RTM at time step max. Action selection proceeds as follows:

(1) We start with an initialization: the set of selected actions Δ_n is initially empty and all new goals at time step n for all resources are equal to the goals at 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 $gmin(n+1,\$r) \leq \mathsf{N} \leq gmax(n+1,\$r)$
    then $gmin(n,\$r) := -\infty$
        $gmax(n,\$r) := +\infty$
    else backtrack to select a new action
else
a is simple-effect producer/consumer and $\$r \neq \$time$:
$gmin(n,\$r) := gmin(n,\$r)\ \overline{\mathsf{OP}}\ \mathsf{N}_{(\mathsf{a})}$
$gmax(n,\$r) := gmax(n,\$r)\ \overline{\mathsf{OP}}\ \mathsf{N}_{(\mathsf{a})}$

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 is not a simple-effect producer/consumer (effect contains linear equation): do nothing

with $\overline{\mathsf{OP}}$ being the inverse operator to OP, i.e., we use $+$ instead of $-$ and vice versa. $\mathsf{N}_{(\mathsf{a})}$ 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 $[150, +\infty)$.

(4) 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. A test against the predeclared interval boundaries is not needed as these are already reflected in the RTM intervals.

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 by actually determining 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 the interval that specifies how much of a resource is initially available is a subinterval of the resource goal for the initial state, i.e., if $[tmin(0,\$r), tmax(0,\$r)] \subseteq [gmin(0,\$r), gmax(0,\$r)]$ for all $\$r$.

In the case of linear equations, the resource effect of an action is ignored because it is unclear what the specific impact on the resource would be until a complete plan is found and the specific value of the dependency variable in a particular state is fixed. Only its resource requirements are considered by the planner because these must be satisfied by any solution plan. The planner could use the worst-case estimate, but this would make the planning process much more complicated if completeness needs to be retained.

If any of the tests succeeds for a minimal set of actions, the planner needs to resolve resource conflicts 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.

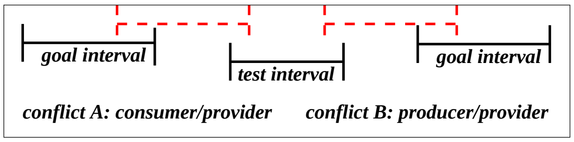

Figure 1: Pushing goal intervals to obtain a non-empty intersection with the test interval.

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. If actions exist in the graph with only resource effects and without logical effects, several copies of the same action can be selected for parallel execution. For example if there is an action

get-sheet-of-paper
```
:p at-paper-tray
:e $sheets  += 1.
```

taking 10 sheets can be done in parallel. This leads to more parallelism in plans and in the case that temporal restrictions have to be met, it can be the only way how a solution to a planning problem becomes possible.

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 $i \geq n$, i.e., a backtracking process tries if adding actions at later levels in the plan and then searching forward again resolves the conflict. The conflicting resource variable is stored in a separate data structure together with the policy to resolve it (i.e., the type of action that has to be considered). When the planner returns to an action level, it does not immediately consider a new set of actions, but tries to extend the current set by one action such that at least one conflict from the agenda is reduced or resolved. Then it searches forward again. This way, all maximal possible sets of non-exclusive actions are considered for a given goal, before another minimal action set is considered.

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 actions levels between max and the level n where the conflict was detected. If no linear equations involving resource dependencies occur in the choosen 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.

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.

In the airplane example, action selection proceeds as follows: We start with the goal intervals for `$time` $(0, 330]$ and `$gas` $(0, 750)$ 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 at time step 3, which leads to the following goals at fact level 3:

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

For `$time` the planner computes $(0-60, 330-60]$ intersected with the resource requirements $(-\infty, +\infty)$ and the RTM interval at fact level 3 $[0, 360]$. For `$gas` the planner computes $(0 + 133.33, 750 + 133.33)$, which gets intersected with the resource requirements $[133.33, +\infty)$ and the RTM interval $[0, 750]$.

To fly Ernie and Scott from Paris to London both have to be on board the aircraft being at Paris. Flying, boarding, and refueling are exclusive, but refueling and boarding can be done in parallel. Let us assume that the planner selects the action **board(Paris)** to have Ernie entering the aircraft, while `boarded(Scott)` and `at-plane(Paris)` are added by a no-op. 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)`
`at-plane(Paris) boarded(Scott)`

No-ops are selected to achieve all subgoals except `at-plane(Paris)` that is achieved by the action **fly(Basel Paris)** which requires 200 amounts of gas and takes 90 minutes time, 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 $[333.33, 750]$.

f 1: `$time ∈ [0, 120] $gas ∈ [333.33,750]`
`checked-in(Ernie) at(Ernie Paris)`
`at-plane(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 `$gas`. Since the goal interval lies "right" of the RTM interval, a producer or provider has to be considered for conflict resolution. The refueling action is the only provider of `$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.

For `$time` we do not need to recompute the new goal because the new producer changes it depending on a linear equation. All goals at time step 0 are supersets of the resource intervals in the initial state and the planner terminates with a plan candidate. But since a non-simple resource effect is used by the plan, this candidate has to be verified by a subsequent symbolic execution phase.

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

5 Verification of Plan Candidates through Symbolic Execution

During graph search and action selection, effects based on linear equations are ignorered, because the specific impact of the effect can only be correctly determined after the exact value of the dependency variable is known. But this becomes possible only after a complete plan has been generated, i.e., as soon as the planner selects an action with a non-simple resource effect, soundness can be affected and a verification of the plan candidate becomes necessary.

To avoid the symbolic execution one had to generate a plan that is sound wrt. best-case and worst-case estimates, i.e., the linear equations had to be equipped with two estimates that have to be considered during planning. For some practical applications this could be useful, but in the general case it would rule out too many solutions.

The symbolic execution takes as input a plan and the initial and goal specification for resource variables and returns true if and only if the plan is a solution to the given planning problem.

In the initial state, an interval is specified for each resource, which can also be infinite. If no interval is given for a resource, it is initialized with the predeclared interval boundaries. Furthermore, the planner tests if all initial intervals are subintervals of the predeclared intervals, otherwise, the planning problem is rejected as inconsistent.

The plan candidate determines which actions are to be executed in each state and with that we know which resource effects will occur because all actions are executable - their logical preconditions are guaranteed to be satisfied in the state and resource effects are unconditional. The resource effects in each action specify which resource variable they affect and a computational instruction says how the resource variable will change. Furthermore, each action has atmost one effect per resource.

Since parallel action sets have to be non-exclusive, each possible linearization has the same impact on resources except on $time for which we assume true parallel execution. This means, times do not sum up, but only the maximal use of time determines the total advance on the time scale that is caused by a parallel set of actions.

The symbolic execution algorithm takes the intervals from the current state and the resource effects of all actions in a parallel action set as input and determines the new resulting state. It is based on the semantics of resources as defined in [6] and tests if each new state is valid and if the final resulting state satisfies the goals.

The symbolic execution itself causes only a marginal computational overhead, but the necessity to construct a complete plan before execution can start leads to a much larger search space. Therefore, resource dependencies should be used rather sparsely in actions whenever possible.

As an example, let us consider the airplane plan with the tighter time requirement `$time` $\leq$ `215` as a goal. This would lead to the following resource goals at each time step:

time	time	gas	actions
0:	$[0, 0]$	$[300, 300]$	board(B) refuel(B)
1:	$[0, 35]$	$[333.33, 750]$	fly(B P)
2:	$[0, 125]$	$[133.33, 750]$	board(P)
3:	$[0, 155]$	$[133.33, 750]$	fly(P L)
4:	$[0, 215]$	$[0, 750]$	

This plan ignores the increase in time caused by the refuel action and is therefore incorrect. Symbolic execution reveals this problem because refueling requires 36 minutes and therefore a violation of the time requirements at fact level 1 and all subsequent levels occurs.

time	time	gas	actions
0:	0	300	board(B) refuel(B)
1:	36	750	fly(B P)
2:	126	550	board(P)
3:	156	550	fly(P L)
4:	216	416.67	

Failures during execution could be analyzed to derive information that guides the subsequent search process for a new plan candidate. However, such failures can have many possible sources and offer a huge range of solution possibilities. It is therefore not obvious how to derive control information. In the example, one can only conclude that refueling this specific amount of gas violates all time requirements in subsequent states. Thus, one could try to reduce gas consumption, or try to refuel in a different way, or relax temporal constraints caused by *flying* actions, etc. All these possibilities are explored during the systematic search of the planning graph and it seems to be impossible to decide in advance which of them should be favored.

6 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 time model is available as for example in [1, 10, 9, 3, 4, 8, 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 test with the current prototype show quite good runtime behavior of our algorithm. For example, the airplane problem as described in this paper is solved by IPP in 0.02 seconds (without any user interaction) compared to 2-3 minutes for ZENO with user-guided goal selection.

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.

Currently, we are concentrating on the development of an efficient termination test for unsolvable problems and techniques to further speed up the planner.

References

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

[2] B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In Hammond [5], pages 243–248.

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

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

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

[6] J. Koehler. Planning under resource constraints in IPP. Technical report, University of Freiburg, 1998. forthcoming.

[7] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel, editor, *Proceedings of the 4th European Conference on Planning*, volume 1348 of *LNAI*, pages 273–285. Springer, 1997.

[8] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 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*, pages 1010–1015. AAAI Press, MIT Press, 1994.

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