

# First Solutions to PDDL+ Planning Problems

Stefan Edelkamp

Institut für Informatik,  
Albert-Ludwigs-Universität,  
Georges-Köhler-Allee, D-79110 Freiburg  
eMail: edelkamp@informatik.uni-freiburg.de

**Abstract.** In this paper we present the design and algorithmic details of a directed search temporal and metric planner to solve benchmark planning problems specified in PDDL+ syntax. The planner handles mixed logical-numerical problems, grounds and groups predicates, simplifies arithmetic trees, and instantiates numerical quantities on the fly. The operators span a possibly infinite state space, which is tackled by guided exploration. The heuristic estimates are inferred by plan relaxation, through pattern databases, or by state-to-goal differences of numerical values.

The planner uses forward state-space search with an A\* search control and is evaluated in problem instances of the two benchmark planning problems *ZenoTravel* and *DesertRats*.

## 1 Introduction

The problem description language PDDL+ is in fact a hierarchy of Lisp-like syntax rules to specify both planning domains and planning problem instances. It revises McDermott's original problem description language PDDL [15]. Levels 1-3 of PDDL+ [9] serve as an officially agreed extension for PDDL to specify benchmark problems instances at the AIPS 2002s international planning competition (IPC). As in current PDDL+ in this paper we stick to full observation and to deterministic actions.

Solving planning problems with numerical preconditions and effects as allowed in Level 2 and 3 problems is undecidable in general [11]. However, the structures of the provided benchmark problems are simpler than the general problem class, so that these problems are in fact solvable. According to the PDDL-hierarchy we indicate three problem classes:

1. Propositional Planning. Strips problems [8] are well-understood, fully propositional and grounding leads to a finite state space, which in fact is a subset of all combinations of propositional atoms. Operators are defined by instantiated precondition and effect lists. Strips problems have been tackled with different planning techniques, most notably by SAT-planning [13], IP-planning [14], CSP-planning [20], BDD-planning [7], graph relaxation [1] and heuristic search planners [2]. The major quality measurements are the

numbers of sequential and parallel steps. ADL generalizations [17] like conditional effects and negative preconditions are more expressive in general, but can usually be resolved during grounding.

2. Numerical Effects. Numerical variables in the effect lists allows to include time and resources. If numerical effects do not bound integral values, infinite state spaces are likely to be generated. However, by assuming finitely many interesting events the problem class becomes tractable and is effectively dealt by schedulers that usually minimize the *make-span* of concurrent actions [10].
3. Numerical Preconditions. We distinguish finite and infinite branching problems. With finite branching, execution time of an action is not parameterized, while with infinite branching, an infinite number of actions can be applied. These problems have ever since been confronted to model checking. Some subclasses of infinite branching problems like timed automata exhibit a finite partitioning through a symbolic representation of states [18]. The absence of partitioning is current research [22].

We concentrate on mixed propositional and numerical planning with finite branching, in which numerical preconditions and effects are determined on the fly. As we will see, this class is inherent to the PDDL+ syntax. Our major objective will be to parse problems instances and to find at least one feasible solution. Optimizing the solution quality with respect to the numerical quantities and concurrency of actions are addressed to future algorithmic design and implementation efforts.

We have structured this paper as follows. Firstly, we present a pre-compiler for PDDL+, that parses and modifies the high level description down to a concise and grounded representation. As an interface for other planners the pre-compiler writes a file for the given planning instances. We take the two published PDDL+ domains *Zeno-Travel* and *Desert-Rat* as example cases. We propose a simple heuristic search planner that solves these planning problems and returns a sequence of timed actions and study the effect of different heuristic estimates, discretization, and branching cuts. Finally, we compare the work with literature and discuss extensions for improved performance and higher specification levels.

## 2 The Zeno-Travel Domain

Zeno-Travel is one of the first two published problems in PDDL+. We have depicted the complete domain specification in Figure 2. It includes durative actions and typing of variables.

Problem Zeno-Travel-1 (cf. Figure 2) asks for a plan to fly certain persons (*dan*, *scott*, and *ernie*) located somewhere on a small map (city-a, city-b, city-c, or city-d) with an aircraft (plane) to their respective destinations. Boarding and debarking takes a constant amount of time. The plane has a determined capacity for fuel. Fuel and time are consumed according to the distances between the cities and with respect to two different speeds. Since fuel can be restored by refueling the aircraft, the total amount of fuel is also maintained as a numerical quantity.

```

(define (domain zeno-travel)
  (:requirements :durative-actions :typing :fluents)
  (:types aircraft person city)
  (:predicates (at ?x - (either person aircraft) ?c - city)
               (in ?p - person ?a - aircraft))
  (:functions (fuel ?a - aircraft)
              (distance ?c1 - city ?c2 - city)
              (slow-speed ?a - aircraft)
              (fast-speed ?a - aircraft)
              (slow-burn ?a - aircraft)
              (fast-burn ?a - aircraft)
              (capacity ?a - aircraft)
              (refuel-rate ?a - aircraft)
              (total-fuel-used)
              (boarding-time)
              (debarking-time))
  (:durative-action board
   :parameters (?p - person ?a - aircraft ?c - city)
   :duration (= ?duration boarding-time)
   :condition (and (at start (at ?p ?c))
                  (over all (at ?a ?c)))
   :effect (and (at start (not (at ?p ?c)))
               (at end (in ?p ?a))))
  (:durative-action debark
   :parameters (?p - person ?a - aircraft ?c - city)
   :duration (= ?duration debarking-time)
   :condition (and (at start (in ?p ?a))
                  (over all (at ?a ?c)))
   :effect (and (at start (not (in ?p ?a)))
               (at end (at ?p ?c))))
  (:durative-action fly
   :parameters (?a - aircraft ?c1 ?c2 - city)
   :duration (= ?duration (/ (distance ?c1 ?c2) (slow-speed ?a)))
   :condition (and (at start (at ?a ?c1))
                  (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))))
   :effect (and (at start (not (at ?a ?c1)))
               (at end (at ?a ?c2))
               (at end (increase total-fuel-used (* (distance ?c1 ?c2) (slow-burn ?a))))
               (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))))
  (:durative-action zoom
   :parameters (?a - aircraft ?c1 ?c2 - city)
   :duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
   :condition (and (at start (at ?a ?c1))
                  (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
   :effect (and (at start (not (at ?a ?c1)))
               (at end (at ?a ?c2))
               (at end (increase total-fuel-used (* (distance ?c1 ?c2) (fast-burn ?a))))
               (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
  (:durative-action refuel
   :parameters (?a - aircraft ?c - city)
   :duration (= ?duration (/ (- (capacity ?a) (fuel ?a)) (refuel-rate ?a)))
   :condition (and (at start (< (fuel ?a) (capacity ?a)))
                  (over all (at ?a ?c)))
   :effect (at end (assign (fuel ?a) (capacity ?a))))

```

Fig. 1. The Zeno-Travel Domain

```

(define (problem zeno-travel-1)
  (:domain zeno-travel)
  (:objects plane - aircraft
            ernie scott dan - person
            city-a city-b city-c city-d - city)
  (:init (= total-fuel-used 0)
        (= debarking-time 20)
        (= boarding-time 30)
        (= (distance city-a city-b) 600)
        (= (distance city-b city-a) 600)
        (= (distance city-b city-c) 800)
        (= (distance city-c city-b) 800)
        (= (distance city-a city-c) 1000)
        (= (distance city-c city-a) 1000)
        (= (distance city-c city-d) 1000)
        (= (distance city-d city-c) 1000)
        (= (fast-speed plane) (/ 600 60))
        (= (slow-speed plane) (/ 400 60))
        (= (fuel plane) 750)
        (= (capacity plane) 750)
        (= (fast-burn plane) (/ 1 2))
        (= (slow-burn plane) (/ 1 3))
        (= (refuel-rate plane) (/ 750 60))
        (at plane city-a)
        (at scott city-a)
        (at dan city-c)
        (at ernie city-c))
  (:goal (and (at ernie city-d)
              (at scott city-d)))
  (:metric minimize total-time))

```

Fig. 2. The Problem Instance Zeno-Travel-1.

### 3 Pre-Compilation

Our simple Lisp parser generates a tree of Lisp entities. It reads the input files and recognizes the domain and problem name. Based on the number of counted objects, a coarse estimate for the grounded predicates and functions is devised. To cope with typing we temporarily assert constant typed predicates to be removed together with other constant predicates in a further pre-compiling step. Thereby, we infer a type hierarchy and an associated mapping of objects to types.

Since in our example problem we have eight objects and the predicates `at` and `in` have two parameters, we reserve  $2 \cdot 8 \cdot 8 = 128$  index positions. Similarly, the functor `distance` consumes 64 indices, while `fuel`, `slow-speed`, `fast-speed`, `slow-burn`, `fast-burn`, `capacity`, and `refuel-rate` each reserves eight index positions. For the quantities `total-fuel-used`, `boarding-time`, `debarking--`

time only a single fact identifier is sufficient. Last but not least we interpret duration as an additional quantity `total-time`.

According to our assumption of finite branching in this phase we interpret each action as in integral entity, so that all timed propositional and numerical preconditions can be merged. Similarly, all effects are merged, independent of their happening. Invariance conditions like `(over all (at ?a ?c))` in the action `board` are included into the precondition set.

Numerical quantities divide into three different sets: formula heads, applied operators, and formula bodies. For preconditions the operator is a comparator (`>`, `>=`, `<`, `<=` and `=`) and for effects it is a modifier (`assign`, `increase` and `decrease`). The formulae bodies are represented as trees with symbolic facts and constants as leaves. Instantiated numerical constants as apparent in the initial and goal states are grounded functions together with the associated values that are interpreted as rational numbers.

However, before instantiating numerical conditions, *fact-space exploration* on the propositional part of the problem is invoked. This is a relaxed exploration of the planning problem to determine a superset of all reachable facts. Algorithmically, a FIFO fact queue is comprised. Successively extracted fact at the front of the queue are matched to the operators. Each time all preconditions of an operator are fulfilled, the resulting atoms according to the positive effect (add) list are determined and inserted to the end of the queue. This allows to distinguish constant from fluent facts, since only the latter are reached by exploration.

For a concise encoding of the propositional part we group fluent facts in sets of mutually exclusive groups, so that each state in the planning space can be expressed as a conjunct of (possibly trivial) facts drawn from each fact group [6]. To allow for a better encoding some predicates like `at` and `in` are merged. In the examples three groups determine the unique position of the persons (one of five) and one group determines the position of the plane (one of four). Therefore  $3 \cdot \lceil \log 5 \rceil + 1 \cdot \lceil \log 4 \rceil = 11$  bits suffice to encode the encountered 19 fluent facts.

Fact-space exploration also determines all grounded operators. Once all preconditions are met and grounded, the symbolic effect-lists are instantiated. In our case we determine 98 instantiated operators, which, by some further simplifications that eliminate duplicates and void operators, are reduced to 43.

Synchronous to fact space exploration of the propositional part of the problem all heads of the numerical formulae in the effect lists are grounded. In the example case only three instantiated formulae are fluent: `fuel plane` with initial value 750 as well as `total-fuel-used` and `total-time` both initialized with zero. All other numerical predicates are in fact constants that can be substituted in the formula-bodies. For example, the numerical effect in `board dan city-a` reduces to `total-time += 30`, while `fly plane city-d city-c` has the following three numerical effects: `total-time += 150`, `total-fuel-used += 1000/3`, and `fuel plane -= 1000/3`. Refueling, however, does not reduce to a single rational number, e.g. the effects in `refuel plane city-d` only simplify to `total-time += (750-fuel plane) / (25/2)` and `fuel plane := 750`. To evaluate the former assignment the numerical variable `total-time` has to be

instantiated *on-the-fly*, during execution. This is due to the fact that the value of the quantity `fuel plane` is not constant and itself changes over time.

Besides initialization of suitable data structures for the exploration, as a byproduct the pre-compiler flushes a file `instance.h` to be used as an interface for other planning engines. Except of grouping of facts, numerical preconditions and more complex numerical formulae, the format matches the one produced by the two-phase planning system TP4 [10].

## 4 Solving Zeno-Travel

For planning we choose the A\* algorithm that instantiates and generates the successor set according to a combined priority of generating path length and heuristic estimate. The important fact is that blind exploration algorithms are likely to become lost in infinite state spaces. In the example, the plane may fly back and forth without making any progress with respect to the goal situation. Therefore, both breadth-first search and depth-first search exhaust time and space resources quite easily. The state description is a possibly compressed bit-state vector for the fluent propositional part and a rational vector for the changeable numerical quantities. Detecting duplicate only on the propositional part prunes away possible solutions, such that in fact the entire state description of propositional and numerical variables is needed for the soundness of any state space search algorithm.

The first idea for guided exploration is to expand states according to their total-time quantity which increases monotonically. Unfortunately, the refuel option leads to small diverse and quantities, such that exploration also fails to terminate. Fortunately, in the given example case the goal is fully propositional and good heuristics for pure propositional planning are known. The most effective ones are the relaxed planning heuristic of the FF planner [12] and the pattern database heuristic as implemented in the Mips planning system [5].

We have integrated both heuristic estimates to our PDDL+ planner. Since pattern databases (PDBs) do exhibit fact group information and do not interfere with node expansions itself, we integrated this heuristic to our forward A\* planning approach. The following solution found with 54 node expansions has length six:

```
[0]- [30] (board scott plane city-a)
[30]-[180] (fly plane city-a city-c)
[180]-[210] (board ernie plane city-c)
[210]-[360] (fly plane city-c city-d)
[360]-[380] (debark scott plane city-d)
[380]-[400] (debark ernie plane city-d)
```

The make-span according to the optimization criterion is definitely not optimal (the last two actions can be executed in parallel), since we assume all actions to be mutually exclusive. However, with this additional assumption the achieved solution quality is considerably good.

For a more difficult instance in which *dan* is additionally required to reach city-a, we used weighted A\* [16] with cost function  $f(u) = g(u) + 2h(u)$ . The planner finds the following solution of length 11 in 85 node expansions:

```
[0]- [30] (board scott plane city-a)
[30]- [180] (fly plane city-a city-c)
[180]- [210] (board dan plane city-c)
[210]- [240] (board ernie plane city-c)
[240]- [390] (fly plane city-c city-d)
[390]- [410] (debark scott plane city-d)
[410]-[1390/3] (refuel plane city-d)
[1390/3]-[1450/3] (debark ernie plane city-d)
[1450/3]-[1900/3] (fly plane city-d city-c)
[1900/3]-[2350/3] (fly plane city-c city-a)
[2350/3]-[2410/3] (debark dan plane city-a)
```

In both cases the running time for finding the solution<sup>1</sup> is by far smaller than one second. The experimental results in the Zeno-Travel domain<sup>2</sup> are summarized in Table 1.

We highlight that (for this small data set) applying the FF heuristic yields even better performance than searching with the PDB heuristic. Furthermore, we observe that enforced hill-climbing as proposed by the FF planning algorithm starts to backtrack early indicating that it is likely to get stuck in dead-end situations.

## 5 Solving Desert-Rat

Due to duration inequalities the other published benchmark domain *Desert-Rat* is a domain with an infinite branching factor. Therefore, a symbolic state representation is required. The idea is to maintain and update a set of constraints for each numeric variable. For some classes of so-called difference constraints a suitable representation as a weighted constraint graph [3]. Moreover, by the technique of shortest-path reduction a unique and reduced normal form can be obtained. We have implemented this constraint network data structure, since this is the main data structure when exploring timed automata as done by the model checker Uppaal [18]. For this to work, all constraints must have the form  $x_i - x_j \leq c$  or  $x_i \leq c$ . For example, the set of constraints  $x_4 - x_0 \leq -1$ ,  $x_3 - x_1 \leq 2$ ,  $x_0 - x_1 \leq 1$ ,  $x_5 - x_2 \leq -8$ ,  $x_1 - x_2 \leq 2$ ,  $x_4 - x_3 \leq 3$ ,  $x_0 - x_3 \leq -4$ ,  $x_1 - x_4 \leq 7$ ,  $x_2 - x_5 \leq 10$ , and  $x_1 - x_5 \leq 5$  has the shortest-path reduction  $x_4 - x_0 \leq -1$ ,  $x_3 - x_1 \leq 2$ ,  $x_5 - x_2 \leq -8$ ,  $x_0 - x_3 \leq -4$ ,  $x_1 - x_4 \leq 7$ ,  $x_2 - x_5 \leq 10$ , and  $x_1 - x_5 \leq 5$ . If the constraint set is over-constraint, the algorithm will determine unsolvability, otherwise a feasible solution is returned.

<sup>1</sup> On a Pentium III/800 PC Linux computer, 512 MByte

<sup>2</sup> Note that the only difference of Zeno-Travel-1 to Zeno-Travel-2 and Zeno-Travel-3 is a varied merit function that has no influence on the above planning algorithm.

<i>Zeno-Travel-1, Complex, PDB-heuristic</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	0.30s	0.28s	0.29s	0.30s
length	6	6	6	7
expansions	510	54	13	10

<i>Zeno-Travel-1, Complex, PDB-heuristic</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	1.83s	0.38s	0.31s	0.30s
length	11	11	11	12
expansions	49,901	2,500	85	17

<i>Zeno-Travel-1, Simple, FF-heuristic</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	0.34s	0.30s	0.31s	0.31s
length	6	6	7	7
expansions	510	8	7	7

<i>Zeno-Travel-1, Complex, FF-heuristic</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	3.10s	0.34s	0.31s	0.31s
length	11	11	12	12
expansions	49,901	255	15	13

**Table 1.** Experimental results in *Desert-Rat*.

Unfortunately, in PDDL+ not all constraints have this simple form for a constraint network, such that we take a different approach and discretize the domain. The operators `drive-out` and `drive-back` are substituted by discrete variants. The simplified domain is depicted in Figure 4 and includes typing and further preconditions to disallow negative numerical values. Character  $X$  has to be substituted by either 5 or 10. It is important to note that the discretization is done by hand, and it seems difficult to infer a general discretization rule.

In the problem instance *Cross-The-Desert* (cf. Figure 3) six supply tanks are available as a fuel resource for one truck to finally reach the goal distance 300 from the base at distance 0. Pre-compiling *Desert-Rat* establishes that the propositional part can be encoded with 7 bits only: six bits determine the position of the supply-tanks (either at a location or in a truck) and one bit determines the emptiness of the truck.

Since the goal state only contains numerical information of the distance to cross the desert, no propositional heuristic can be devised. Therefore, the difference of the distance values in the goal and the current state is aimed as a heuristic estimator function.

There are two subtle problems to be resolved. As in the example problem with a goal `distance truck` in miles, numbers can be arbitrary large, failing to estimate the solution length in the number of applied operators. This can

```

(define (problem cross-the-desert)
  (:domain desert-rat)
  (:objects truck - vehicle f1 f2 f3 f4 f5 f6 - supply-tank)
  (:init (= total-fuel-consumed 0)
         (= (distance truck) 0) (= (fuel truck) 10)
         (= (capacity truck) 10) (= (frate truck) 1)
         (= (speed truck) 20) (empty truck)
         (= d5 5) (= d10 10)
         (onground f1) (= (content f1) 10) (= (at f1) 0)
         (onground f2) (= (content f2) 10) (= (at f2) 0)
         (onground f3) (= (content f3) 10) (= (at f3) 0)
         (onground f4) (= (content f4) 10) (= (at f4) 0)
         (onground f5) (= (content f5) 10) (= (at f5) 0)
         (onground f6) (= (content f6) 10) (= (at f6) 0))
  (:goal (= (distance truck) 300))
  (:metric minimize total-fuel-consumed))

```

**Fig. 3.** The problem instance *Cross-The-Desert*.

be dealt with normalisation of the difference values. In the example `distance truck` is increased and decreased by the values 100 and 200. Hence, dividing the lack of distance between the current state and the goal state by 200 yields an admissible lower bound. For this case the maximal change in value can be detected by analysis of the effect lists, that are all constant.

However, for the `at` function a more elaborated reasoning has to be performed. For all supply-tanks the numerical resource variable `at` is assigned to the current `distance truck` value. Hence the maximal difference value for `distance truck` determines the maximal difference value for `at`. In a bootstrapping manner we analyze the numerical effects of any given resource. A vector of all detected maximal difference values is maintained and the operators are instantiated with the current vector. In our implementation we neglect numerical preconditions to keep the maximal difference vector exploration as simple as possible. Simplification is mandatory since in complexity theory terms the task to determine the exact difference values is at least as hard as PDDL+ planning itself.

With the domain specification of Figure 4 first solutions to the *Desert-Rat* domain have been found. In the experiments we varied the distance to be traversed. Both *Desert-Rat* problems with distance 300 and 400 are easy to solve. In fact the solutions differ by the last operator only.

[0]- [0] (load truck f1)	[0]- [0] (load truck f1)
[0]-[10] (drive-out truck)	[0]-[10] (drive-out truck)
[10]-[10] (unload truck f1)	[10]-[10] (unload truck f1)
[10]-[10] (fill-up truck f1)	[10]-[10] (fill-up truck f1)
[10]-[15] (drive-out truck)	[10]-[15] (drive-out truck)

With  $A^*$  and the above heuristic exploration require only 31 node expansions. The distance 500 problem is more difficult and requires at least 12 steps to solve.

```

(define (domain desert-rat)
  (:requirements :typing :fluents :durative-actions :duration-equalities)
  (:types vehicle supply-tank)
  (:predicates
    (empty ?t - vehicle) (onground ?f - supply-tank)
    (in ?f - supply-tank ?t - vehicle) (static ?t - vehicle))
  (:functions
    (distance ?t - vehicle) (at ?f - supply-tank) (content ?f - supply-tank)
    (fuel ?t - vehicle) (capacity ?t - vehicle)
    (speed ?t - vehicle) (frate ?t - vehicle)
    (durationX) (total-fuel-consumed))
  (:durative-action drive-out
    :parameters (?t - vehicle)
    :duration (= ?duration durationX)
    :condition (and (at start (>= (fuel ?t) (* durationX (frate ?t))))
      (at start (static ?t)))
    :effect (and (at end (static ?t))
      (at end (increase (distance ?t) (* durationX (speed ?t))))
      (at end (decrease (fuel ?t) (* durationX (frate ?t))))
      (at end (increase total-fuel-consumed (* durationX (frate ?t))))))
  (:durative-action drive-back
    :parameters (?t - vehicle)
    :duration (= ?duration durationX)
    :condition (and (at start (>= (fuel ?t) (* durationX (frate ?t))))
      (at start (static ?t)))
    :effect (and (at end (static ?t))
      (at end (decrease (distance ?t) (* durationX (speed ?t))))
      (at end (decrease (fuel ?t) (* durationX (frate ?t))))
      (at end (increase total-fuel-consumed (* durationX (frate ?t))))))
  (:action load
    :parameters (?t - vehicle ?f - supply-tank)
    :precondition (and (= (distance ?t) (at ?f))
      (static ?t) (onground ?f) (empty ?t))
    :effect (and (not (onground ?f)) (not (empty ?t)) (in ?f ?t)))
  (:action unload
    :parameters (?t - vehicle ?f - supply-tank)
    :precondition (and (static ?t) (in ?f ?t))
    :effect (and (onground ?f) (empty ?t) (not (in ?f ?t)) (assign (at ?f) (distance ?t))))
  (:action fill-up
    :parameters (?t - vehicle ?f - supply-tank)
    :precondition (and (< (fuel ?t) (capacity ?t))
      (= (distance ?t) (at ?f))
      (> (content ?f) (- (capacity ?t) (fuel ?t)))
      (< (capacity ?t) (+ (content ?f) (fuel ?t)))
      (static ?t) (onground ?f))
    :effect (and (assign (fuel ?t) (capacity ?t))
      (decrease (content ?f) (- (capacity ?t) (fuel ?t))))
  (:action refuel
    :parameters (?t - vehicle ?f - supply-tank)
    :precondition (and (< (fuel ?t) (capacity ?t))
      (> (content ?f) 0)
      (= (distance ?t) (at ?f))
      (<= (fuel ?t) (- (capacity ?t) (content ?f)))
      (static ?t) (onground ?f))
    :effect (and (increase (fuel ?t) (content ?f))
      (assign (content ?f) 0)))

```

Fig. 4. Discretized Desert Rats.

```

[0]- [0] (load truck f1)
[0]- [5] (drive-out truck)
[5]- [5] (unload truck f1)
[5]-[10] (drive-back truck)
[10]-[10] (load truck f2)
[10]-[10] (fill-up truck f3)
[10]-[15] (drive-out truck)
[15]-[15] (fill-up truck f1)
[15]-[25] (drive-out truck)
[25]-[25] (unload truck f2)
[25]-[25] (fill-up truck f2)
[25]-[35] (drive-out truck)

```

The number of expansions with A\* is 14,928. Even if supplied with a heuristic estimate the *Desert-Rat* problems with distance 600 is very complex for the search engine.

The following solution of length 24 is the only one we found with a full successor set: Weighted A\*,  $f = g + 10h$ , expands 385,418 node in 22.52s.

```

[0]- [0] (load truck f5)      [25]-[25] (fill-up truck f5)
[0]- [5] (drive-out truck)   [25]-[30] (drive-out truck)
[5]- [5] (unload truck f5)   [30]-[30] (unload truck f3)
[5]-[10] (drive-back truck)  [30]-[35] (drive-back truck)
[10]-[10] (load truck f6)    [35]-[35] (load truck f6)
[10]-[10] (fill-up truck f2) [35]-[35] (refuel truck f5)
[10]-[15] (drive-out truck)  [35]-[40] (drive-out truck)
[15]-[15] (unload truck f6)  [40]-[40] (fill-up truck f3)
[15]-[20] (drive-back truck) [40]-[50] (drive-out truck)
[20]-[20] (load truck f3)    [50]-[50] (unload truck f6)
[20]-[20] (fill-up truck f1) [50]-[50] (fill-up truck f6)
[20]-[25] (drive-out truck)  [50]-[60] (drive-out truck)

```

The core problem is the combinatorial explosion according to the inherent symmetry of the problem with respect to the supply-tanks. This problem can effectively tackled by generating only one successor state to the action schemas `load` and `unload`. With A\*, our planner expands 35,998 nodes to find a solution of length 21. All results of the experiments in the *Desert-Rat* domain are depicted in Table 2.

## 6 Conclusion

Due to the international planning competition IPC on AIPS 2002 there is a rising scientific interest in PDDL+. In this paper we give a preliminary report on parsing and solving PDDL+ problems. Up to the authors knowledge the presented pre-compiler and planner is the first one that can cope with PDDL+ expressiveness.

<i>Cross-the-Desert, Distance 300, No Cuts</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	0.36s	0.34s	0.33s	0.32s
length	5	5	5	5
expansions	95	31	23	6

<i>Cross-the-Desert, Distance 300, Cuts</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	0.35s	0.35s	0.35s	0.35s
length	5	5	5	5
expansions	20	11	8	6

<i>Cross-the-Desert, Distance 500, No Cuts</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	19.83s	13.51s	2.82s	2.29s
length	12	12	12	12
expansions	16,556	14,928	7,781	11,718

<i>Cross-the-Desert, Distance 500, Cuts</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	0.36s	0.44s	0.39s	0.41s
length	12	12	12	12
expansions	1,183	1,322	660	957

<i>Cross-the-Desert, Distance 600, Cuts</i>				
$f =$	$g$	$g + h$	$g + 2 \cdot h$	$g + 10 \cdot h$
time	1.83s	1.16s	2.51s	1.26s
length	21	21	21	24
expansions	50,732	35,998	24,457	14,679

**Table 2.** Experimental results in *Desert-Rat*.

Since the language is high-level and not decidable in general, pre-compiling is at the core of any planner to solve PDDL+ problems. So far, the implementation efforts for parsing the domain into suitable data structures exceeded the implementation time for the heuristic search planner. We altered and integrated propositional search strategies based on an extended state description. Our planner is written in the object-oriented language C++, and by class inheritance most of the code for memorization, heuristic estimates and exploration are shared.

Essentially planning with numerical quantities is planning with time and resources. A limitation of the current approach is a failure to handle time properly. The plans are all sequential and time is irrelevant to the planning process. Our proposed solution to this problem [4] of minimizing the make-span uses critical path analysis for fast and optimal scheduling established sequential plans, and an any-time wrapper to improve the quality of the obtained solution. If the

planner terminates, which is necessarily true in finite state spaces, it outputs the optimal concurrent plan. The difficulty of larger numeric problems is beginning to take a hold in *DesertRats*. This is due to numbers as much as failure to account adequately for symmetry a core aspect for future research.

Some other planners like TP4 [10] are in fact scheduling systems based on the pre-compiled and grounded problem instances. The approach is more expressive as the classical planners Temporal Graphplan (TP) [21] and Greedy Regression Table (GRT)-R [19] and quite fast in practice at least for small-sized problems. Our planner has two distinctive advantages to TP4: it handles numerical pre-conditions and instantiate numerical conditions on the fly.

A further step is to enlarge the problem class. However, it is not yet clear if the benchmark domains yield finite partitionings as with difference conditions in Uppaal. Nevertheless constraint satisfaction is a very elegant and general concept, that can lead to good performance. But more benchmark domains are needed to qualify the performance of the described system and to design suitable algorithms for a larger problem class.

*Acknowledgments* The author would like to thank M. Fox and M. Helmert for helpful discussions concerning this research. The work is supported by DFG in a project entitled *Heuristic Search and its Application to Protocol Verification*

## References

1. A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1636–1642, 1995.
2. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
4. S. Edelkamp. Critical-path scheduling for PDDL+ planning problems. In *Artificial Intelligence Planning and Scheduling (AIPS)*, 2001. Submitted.
5. S. Edelkamp. Planning with pattern databases. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science. Springer, 2001. 13-24.
6. S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pages 135–147. Springer, 1999.
7. S. Edelkamp and M. Helmert. The model checking integrated planning system MIPS. *AI-Magazine*, pages 67–71, 2001.
8. R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
9. M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK, 2001.
10. P. Haslum and H. Geffner. Heuristic planning with time and resources. In *European Conference on Planning (ECP)*, pages 121–132, 2001.
11. M. Helmert. Decidability and undecidability results for planning with numerical state variables, 2001. Submitted to AIPS 2002.
12. J. Hoffmann and B. Nebel. Fast plan generation through heuristic search. *Artificial Intelligence Research*, 14:253–302, 2001.

13. H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI)*, pages 1194–1201, 1996.
14. H. Kautz and J. Walser. State-space planning by integer optimization. In *National Conference on Artificial Intelligence (AAAI)*, 1999.
15. D. McDermott. The 1998 AI Planning Competition. *AI Magazine*, 21(2), 2000.
16. J. Pearl. *Heuristics*. Addison-Wesley, 1985.
17. E. Pednould. ADL: Exploring the middleground between Strips and situation calculus. In *Knowledge Representation (KR)*, pages 324–332. Morgan Kaufman, 1989.
18. P. Pettersson. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. PhD thesis, Department of Computer Systems, Uppsala University, 1999.
19. I. Refanidis and I. Vlahavas. Heuristic planning with resources. In *European Conference on Planning (ECAI)*, pages 521–525, 2000.
20. J. Rintanen and H. Jungholt. Numeric state variables in constraint-based planning. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pages 109–121. Springer, 1999.
21. D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 326–333, 1999.
22. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 88–97. Springer, 1998.