Taming Numbers and Durations in the Model Checking Integrated Planning System

Stefan Edelkamp
Institut für Informatik, Georges-Köhler-Allee, Gebäude 51,
Albert-Ludwigs-Universität, 79110 Freiburg, Germany
eMail: edekamp@informatik.uni-freiburg.de

September 20, 2002

Abstract

The Model Checking Integrated Planning System (MIPS) has shown distinguished performance in the second and third international planning competitions. With its object-oriented framework architecture MIPS clearly separates the portfolio of explicit and symbolic heuristic search exploration algorithms from different on-line and off-line computed estimates and from the grounded planning problem representation.

In 2002, the domain description language for the benchmark problems has been extended from pure propositional planning to include rational state resources, action durations, and plan quality objective functions. MIPS has been the only system that produced plans in each track of every benchmark domain. This article presents and analyzes the algorithmic novelties necessary to tackle the new layers of expressiveness.

The planner extensions include critical path analysis of sequentially generated plans to generate optimal parallel plans. The linear time algorithm bypasses known NP hardness results for partial ordering with mutual exclusion by scheduling plans with respect to the set of actions *and* the imposed causal structure. To improve exploration guidance approximate plans are scheduled for each encountered planning state.

One major strength of MIPS is its static analysis phase that grounds and simplifies parameterized predicates, functions and operators, that infers single-valued invariances to minimize the state description length, and that detects symmetries of domain objects. The aspect of object symmetry is analyzed in detail.

The paper shows how temporal plans of any planner can be visualized in Gannt-chart format in a client-server architecture. The frontend turns also be appropriate for concise domain visualization.

Contents

1 Introduction

The Model Checking Integrated Planning System MIPS has participated twice in the international planning competition: in the second planning competition at AIPS-2000 in Beckenridge (USA) and in the third planning competition at AIPS-2002 in Toulouse (France). As the name indicates, the MIPS project targets the integration of model checking techniques into a domain-independent action planner.

Model checking (Clarke, Grumberg, & Peled, 1999) is the automated process to verify if a formal model of a system satisfies an specified temporal property or not. As an illustrative example, take an elevator control system together with a correctness property that requires an elevator to eventually stop on every call of a passenger or that guarantees that the door is closed, while the elevator is moving.

Although the success in checking correctness is limited, model checkers found many subtle errors in current hardware and software designs. Models often consists of many concurrent subsystems. Their combination is either synchronous, as often met in hardware design verification, or asynchronous, as frequently given in communication and security protocols, or in multi-threaded programming languages like Java.

Exploration of model checking domains spans very large spaces of all reachable system states. This effect is usually denoted as the *state explosion problem*, even if the sets of generated states rather than the states themselves grow that quickly.

An error that shows a safety property violation, like a deadlock or a failed assertion, corresponds to one of a set of target nodes in the state space graph. Roughly speaking, *something bad has occured*. A liveness property violation refers to a (seeded) cycle in the graph. Roughly speaking, *something good will never occur*. For the case of the elevator example, eventually reaching a target state where a request button was pressed is a liveness property, while certifying closed doors refers to a safety property.

In this paper we refer to safety properties only, since goal achievement in traditional and competition planning problems have yet not been extended with temporal properties. However, temporally extended goals are of increasing research interests (Kabanza, Barbeau, & St-Denis, 1997; Pistore & Traverso, 2001; Lago, Pistore, & Traverso, 2002).

The two main validation processes in model checking are explicit and symbolic search. In explicit-state model checking each state refers to a fixed memory location and the state space graph is implicitly generated by successive expansions of state.

In symbolic model checking (McMillan, 1993; Clarke, McMillan, Dill, & Hwang, 1992), fixed-length binary encodings of states are usually seen as mandatory, so that each state can be represented by its characteristic Boolean function. The function evaluates to true if and only if all state variables are assigned to according bit values. Sets of states are expressed by the disjunct of the individual characteristic functions. On the other hand satisfiability and uniqueness of Boolean formulae is NP hard.

The unique symbolic representation of sets of states as Boolean formulae through binary decision diagrams (BDDs) (Bryant, 1992) is often much smaller than the explicit one. BDDs are (ordered) read-once branching programs with nodes corresponding to variables, edges corresponding to variable outcomes, and each path corresponding to an assignment to the variables with the resulting evaluation at the leaves. One reason of the succinctness of BDDs is that directed acyclic graphs may express exponentially many paths. Since states are encoded in binary, the transition relation is defined on two state variable sets. It evaluates to true, if and only if an operator exists that transforms a state into a valid successor. In some sense, BDDs exploit regularities of the state set and often appear better suited to regular hardware systems, in contrast to many software system that inherit a highly asynchronous and irregular structure, so that the straight use BDD with their fixed variable ordering is probably not flexible enough.

For symbolic exploration a set of states is combined with the transition relation to compute the set of all possible successor states, i.e. the image. Starting with the initial state, iteration

of image computations eventually explores the entire reachable state space. To improve the efficiency of image computations, transition relations are often provided in partitioned form.

The correspondence of action planning and model checking can be roughly characterized as follows. Similar to model checkers, action planners implicitly generate large state spaces, and both exploration approaches base on applying parameterized operators to the current state. States in model checking and in planning problems are both labeled by (propositional state) predicates. The satisfaction of a specified property on the one side and the arrival at a certain goal state on the other, leads to a slight difference in the according search objective. With this respect, the goal in action planning is a safety error and the corresponding (error) trail is interpreted as a plan. In the elevator example, the goal of a planning task is to reach a state, in which the doors are open and the elevator is moving. For a formal treatment on the embedding of planning problems into model checking terminology, we refer the reader to (Giunchiglia & Traverso, 1999).

Model checkers perform either symbolic or explicit exploration. To the contrary MIPS features both and allows to combines symbolic and explicit search planning. It applies heuristic search; a search acceleration technique that has let to considerable gains in both communities. In the last few years, heuristic search planners frequently outperform other domain-independent planning approaches, e.g. (Hoffmann & Nebel, 2001), and heuristic search model checkers turn out to significantly improve state-of-the-art, e.g. (Edelkamp, Leue, & Lluch-Lafuente, 2002).

Including resource variables (like the fuel level of a vehicle or the distance between locations) and action duration (i.e. the time passed during execution of the planning operator) are relatively new aspects for action planning, at least in form of an accepted domain description accessible for competitive planning (Fox & Long, 2001). The competition input format PDDL2.1 is not restricted to variables of finite domain, but also includes specification of rational (floating-point) variables in both precondition and effects. Similar to a set of atoms described by a propositional predicate, a set of numerical quantities can be described by a set of parameters. Through the notation of PDDL2.1, we refer to parameterized numerical quantities as functions. For example, the fuel level might be parameterized by the vehicle that is present in the problem instance description.

In the 2002 competition, domains were provided in different tracks according to different layers of language expressiveness: *i*) pure propositional planning, *ii*) planning with numerical resources, *iii*) planning with numerical resources and constant action duration, *iv*) planning with numerical resources and variable action duration, and, in some cases, *v*) more complex problems usually combining time and numbers in more interesting ways. MIPS competed as a fully automated system and performed remarkably well in all five tracks; it solved a high number of problems and was the only system that produced solutions in each track of every benchmark domain.

In this paper the main algorithmic aspects to *tame* rational numbers, objective functions, and action duration are described. The article is structured as follows. First, we recall the development of the MIPS system and assert its main contributions to the planning community. Then we address the object-oriented heuristic search framework architecture of the system. Subsequently, we fix some terminology that allows to give a formal definition of the syntax and the semantics of a grounded mixed numerical and propositional planning problem instance.

We then introduce the core contributions: critical path scheduling for concurrent plans, and efficient methods for detecting and using symmetry cuts. PERT scheduling produces optimal parallel plans given a sequence of operators and a precedence relation among them in linear time. The paper discusses pruning anomalies and handling of different optimization criteria. We analyze the correctness and efficiency of symmetry detection in detail. Afterwards, a TCP/IP client-server visualization system for sequential and temporal plans is presented. The article closes with related work and concluding remarks.

2 The Development of MIPS

The competing versions of MIPS refer to initial findings (Edelkamp & Reffel, 1999) of heuristic symbolic exploration of planning domains with the μcke model checker (Biere, 1997) that already lead to good performance in puzzle solving (Edelkamp & Reffel, 1998) and in hardware verification (Reffel & Edelkamp, 1999). For general propositional planning, our concise BDD library *StaticBdd*[1] has been used.

During the implementation process we changed the BDD representation to improve performance mainly for small planning examples and chose the public domain `c++` BDD package Buddy (Lind-Nielsen, 1999). In the beginning of the project the variable encodings were provided by hand, while the representation of all possible operator descriptions were established by enumerating all possible parameter instances. Once the encoding and transition relation were fixed, symbolic exploration in form of a reachability analysis of the state-space could be executed. At that time, we were not aware of any other work in BDD-based planning like (Cimatti, Giunchiglia, Giunchiglia, & Traverso, 1997), which is probably the first link to planning via (symbolic) model checking.

Since the above approach was criticized not to be fully automated, we subsequently developed a parser and a static analyzer to cluster atoms into groups in order to minimize the length of the state encoding (Edelkamp & Helmert, 1999). The outcome of the analyzer allowed to specify states and transition functions in Boolean terms, which in turn were included in a bidirectional BDD exploration and solution extraction procedure. In the end, MIPS was the first automated planning system based on symbolic model checking.

In the second international planning competition MIPS (Edelkamp & Helmert, 2001) could handle the STRIPS (Fikes & Nilsson, 1971) subset of the PDDL language (McDermott, 2000) and some additional features from ADL (Pednould, 1989), namely negative preconditions and (universal) conditional effects. MIPS was one of five planning systems to be awarded for "Distinguished Performance" in the fully automated track. The competition version (Edelkamp & Helmert, 2000) already included explicit heuristic search algorithms based on a bit-vector state representation and the relaxed planning heuristic (RPH) (Hoffmann & Nebel, 2001) and symbolic heuristic search based on the HSP-Heuristic (Bonet & Geffner, 2001) and a one-to-one atom RPH-derivate. However, at the end we used breadth-first bi-directional symbolic search in each case the single state heuristic searcher got stuck in its exploration.

In between the planning competitions, explicit (Edelkamp, 2001c) and symbolic pattern databases (Edelkamp, 2002b) were proposed as off-line generated estimators referring to completely explored problem abstractions. Roughly speaking, pattern database abstractions slice the state vector of fluent facts into pieces and adjusts the operators accordingly. The completly explored subspaces then serve as admissible estimate for the overall search and are competetive with the relaxed planning heuristic.

For the 2002's international planning competition new levels of the planning domain description language (Fox & Long, 2001) have been designed to specify problems that include actions with durations and resources. The agreed input language definition is referred to as PDDL 2.1. While Level 1 considers pure propositional planning, Level 2 also includes numerical resources and objective functions to be minimized, and Level 3 additionally allows to specify actions with durations. Consequently, MIPS[2] has been extended to cope with these new forms of expressiveness.

In (Edelkamp, 2001b) first results of MIPS in planning PDDL 2.1 problems are presented. The preliminary treatment exemplifies the parsing process in two simple benchmark domains. Moreover, propositional heuristics and manual branching cuts were applied to accelerate sequential plan generation. This work was extended in (Edelkamp, 2002a), where two approximate exploration techniques to bound and to fix numerical domains, first results on symmetry detec-

[1] See `http://www.informatik.uni-freiburg.de/~edelkamp/StaticBdd`

[2] A recent version of MIPS is available in source code at `www.informatik.uni-freiburg.de/~edelkamp`

tion based on fact groups and critical path scheduling, an any-time wrapper to produce optimal plans and a numerical extension to RPH were presented. Enumerating variable domains and the any-time wrapper were excluded from the competition version of MIPS because of their unpredictable impact on the planner performance.

Our approach to extend RPH with numerical information establishes plans even in challenging numerical domains like *Settlers* and was developed independently from Hoffmann's work on his competing planner *Metric-FF*. Since his planner appears to be more general in its parsing process to generate monotone numerical quantities for the relaxation, we omitted the algorithmic issues for this aspect from this manuscript. The reader is refered to (Hoffmann, 2002a) for further information on this important issue of relaxed plan generation.

Although possible and plausible, in the competition, (S)PDBs estimates were finally not included for plan generation in MIPS, since the integration of numerical state facets and the retrieval of the corresponding relaxed plan operators had not been finished. Hence, the applied heuristic search engine at least in the competition version of MIPS relates to a numerical relaxed plan generator with important pre- and postprocessing aspects.

Hence, we selected the main contributions of this paper to include the following aspects:

- the formal definition of grounded propositional and numerical planning and an index scheme for grounding predicate, functions, and actions;
- the object-oriented framework architecture to choose and combine different heuristics with different search algorithms and storage structures;
- the static analyzer that applies efficient fact-space exploration to distinguish constant from variable atoms and resource variables, that clusters facts into groups and that infers static object symmetries;
- different pruning methods, especially dynamic symmetry detection, hash and transposition cuts, and different strategies for optimizing objective functions and further implementation tricks that made the system efficient;
- a throughout study of dynamic object symmetries, their time and space complexities and a possible trade-off as implemented in MIPS;
- optimal temporal planning enumeration algorithms based on a precedence ralation and PERT scheduling of sequentially generated plans together with a concise analysis of correctness and optimality;
- the integration of PERT scheduling already in the heuristic estimate to guide the search favoring states with smaller parallel plan length;
- the intermediate format of grounded and simplified planning domain instances to serve as an interface for other planners;
- a client-server system for visualization including a wrapper for temporal plans to be presented in Gannt-chart format, and a domain-dependent frontend for executing sequential plans.

3 Terminology

Our running example is the following instance to of a simple PDDL 2.1 problem in *Zeno-Travel* and illustrated in Figure 1. The initial configuration is drawn to the left of the figure and the goal configuration to its right. Some global and local numerical variable assignment are not shown.

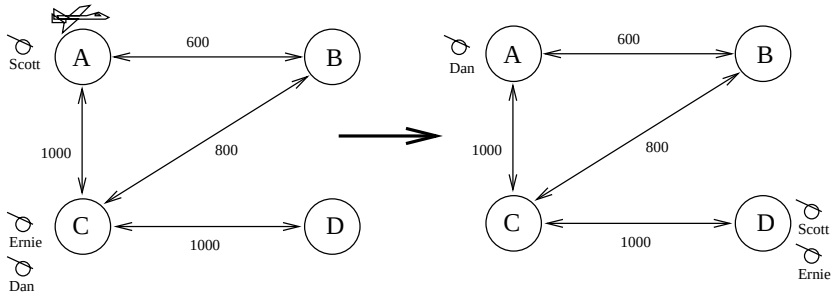

Figure 1: An Instance to the *Zeno-Travel* Domain: Start State (left) and Goal State (right).

In Figures 2 and 3 we provide the according textual domain and problem specification[3]. The instance asks for a temporal plan to fly passengers (`dan`, `scott`, and `ernie`) located somewhere on a small map (including the cities `city-a`, `city-b`, `city-c`, and `city-d`) with an aircraft (`plane`) to their respective target destinations. Boarding and debarking takes a constant amount of time. The plane has a determined capacity of fuel. Fuel and time are consumed according to the distances between the cities and with respect to two different travel speeds. Since fuel can be restored by refueling the aircraft, the total amount of fuel is also maintained as a numerical quantity.

3.1 Sets and Indices

Table 1 displays the basic terminology for sets used in this paper. As most currently successful planning system, MIPS grounds parameterized information present in the domain description.

Set	Descriptor	Example(s)
$\mathcal{OBJ}$	objects	`dan`, `city-a`, `plane`, ...
$\mathcal{TYPE}$	object types	`aircraft`, `person`, ...
$\mathcal{PRED}$	predicates	`(at ?a ?c)`, `(in ?p ?a)`, ...
$\mathcal{FUNC}$	numerical functions	`(fuel ?a)`, `(total-time)`, ...
$\mathcal{ACT}$	parameterized actions	`(board ?a ?p)`, `(refuel ?a)`, ...
$\mathcal{IACT}$	instantiated actions	`(board plane scott)`, ...
$\mathcal{O} \subseteq \mathcal{IACT}$	fluent operators	`(board plane scott)`, ...
$\mathcal{IPRED}$	instantiated predicates	`(at plane city-b)`, ...
$\mathcal{F} \subseteq \mathcal{IPRED}$	fluents	`(at plane city-b)`, ...
$\mathcal{IFUNC}$	instantiated funtions	`(distance city-a city-b)`, ...
$\mathcal{V} \subseteq \mathcal{IFUNC}$	variables	`(fuel plane)`, `(total-time)`, ...

Table 1: Basic Set Definitions.

For all sets we infer a suitable array embedding, indicated by a mapping ϕ from this set to a finite domain and vice versa. This embedding is important to deal with unique identifiers of entities instead of their textual or internal representation. The arrays containing the corresponding information can then be accessed in constant time. Almost all planners that perform grounding prior to the search address instantiations by identifiers.

For sets that occur in the domain or problem specification without any parameterization like $\mathcal{CONST}, \mathcal{PRED}, \mathcal{FUNC}, \mathcal{ACT}, \mathcal{TYPE}, \mathcal{ACT}$, and $\mathcal{OBJ}$, the index ϕ refers to the position of

[3] `[...]` denotes that source fragments were omitted for the sake of brevity. In the given example these are the action definitions for debarking a passenger and flying an airplane..

```
(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 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))))
)
```

Figure 2: Zeno-Travel Domain Description in PDDL2.1.

occurrence. Let $k(p)$, $k(f)$, and $k(a)$ denote the arity (the number of parameters) of predicate $p \in \mathcal{PRED}$, function $f \in \mathcal{FUNC}$, and $a \in \mathcal{ACT}$, respectively. The index for an instantiated predicate $(p\ o_1 \ldots o_{k(p)}) \in \mathcal{IPRED}$ is computed as

$$\phi((p\ o_1 \ldots o_{k(p)})) = \psi(p) + \sum_{i=1}^{k(p)} \phi(o_i)|\mathcal{OBJ}|^{i-1},$$

where $\psi(p) = \sum_{i=1}^{\phi(p)-1} |\mathcal{OBJ}|^{k(p_i)}$ is the offset of predicate $p \in \mathcal{PRED}$ and $|\mathcal{OBJ}|$ is the cardinality of set $\mathcal{OBJ}$. Taking $|\mathcal{OBJ}|$ as the radix is a rather coarse value for all parameter instantiations; one could refine the index by using parameter type information.

Indices for instantiated functions $(f\ o_1 \ldots o_{k(f)}) \in \mathcal{IFUNC}$ are determined analogously. Instantiated actions $a \in \mathcal{IACT}$ with parameters $p_1, \ldots, p_{k(a)}$ are consequently addressed by the following index

```
(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 dan city-a) (at ernie city-d) (at scott city-d)))
    (:metric minimize total-time)
)
```

Figure 3: Zeno-Travel Problem Instance.

$$\phi((a\ p_1 \dots p_{k(a)})) = \psi(a) + \sum_{i=1}^{k(a)} \phi(p_i)|\mathcal{OBJ}|^{i-1}.$$

After static analysis has established a superset of all occurring fluents $\mathcal{F}$, operators $\mathcal{O}$ and variables $\mathcal{V}$, in MIPS the index range is reduced to a minimum, thereby refining ϕ to ϕ'. In the following we keep ϕ as a descriptor, and assume that $\phi(p) \in \{1, \dots, |\mathcal{P}|\}$, $\phi(f) \in \{1, \dots, |\mathcal{V}|\}$, and $\phi(a) \in \{1, \dots, |\mathcal{O}|\}$.

In the following we first give the formal description of a grounded planning problem and then turn to the static analyzer that infers the according and supplementary information.

3.2 Grounded Planning Problem Instances

As many other planners MIPS refers to grounded planning problem representations.

Definition 1 *(Grounded Planning Instance) A* grounded planning instance *is a quadruple* $\mathcal{P} = \langle \mathcal{S}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$*, where* $\mathcal{S}$ *is the set of planning states,* $\mathcal{I} \in \mathcal{S}$ *is the initial state,* $\mathcal{G} \subseteq \mathcal{S}$ *is the set of goal states. In mixed propositional and numerical planning problem the state space* $\mathcal{S}$ *is given by*

$$\mathcal{S} \subseteq 2^{\mathcal{F}} \times I\!R^{|\mathcal{V}|},$$

where $2^{\mathcal{F}}$ *is the power set of* $\mathcal{F}$*. Therefore, a state* $S \in \mathcal{S}$ *is a pair* (S_p, S_n) *with propositional part* $S_p \in 2^{\mathcal{F}}$ *and numerical part* $S_n \in I\!R^{|\mathcal{V}|}$*.*

For the sake of brevity, we assume the operators to be in *normal form*, by means that propositional parts (preconditions and effects) satisfy standard STRIPS notation (Fikes & Nilsson, 1971) and numerical parts are given in form of arithmetic trees t taken from the set of all trees T with arithmetic operations in the nodes and numerical variables and evaluated constants in the leaves. With $\mathit{LeafVariables}(t)$, $t \in T$, we denote the set of all leaf variables in the tree t. However, there is no fundamental difference to more general preconditions and effects representations. The current implementation in MIPS takes a generic precondition tree, thereby including comparison symbols, logical operators (in the nodes) and arithmetic subtrees.

Definition 2 *(Syntax of Grounded Planning Operator) An operator $o \in \mathcal{O}$ in normal form $o = (\alpha, \beta, \gamma, \delta)$ has propositional preconditions $\alpha \subseteq \mathcal{F}$, propositional effects $\beta = (\beta_a, \beta_d) \subseteq \mathcal{F}^2$, numerical preconditions γ, and numerical effects δ. A* numerical precondition *$c \in \gamma$ is a triple $c = (h_c, \otimes, t_c)$, where $h_c \in \mathcal{V}$, $\otimes \in \{\leq, <, =, >, \geq\}$, and $t_c \in T$. A* numerical effect *$m \in \delta$ is a triple $m = (h_m, \oplus, t_m)$, where $h_m \in \mathcal{V}$, $\oplus \in \{\leftarrow, \uparrow, \downarrow\}$ and $t_m \in T$.*

Obviously, $\otimes \in \{\leq, <, =, >, \geq\}$ represents the associated comparison relation, while $\leftarrow$ denotes an assigment to a variable, while $\uparrow$ and $\downarrow$ indicate a respective increase or decrease operation to it. This allows to formalize the application of planning operators to a given state.

Definition 3 *(Semantics of Grounded Planning Operator Application) An operator $o = (\alpha, \beta, \gamma, \delta) \in \mathcal{O}$ applied to a state $S = (S_p, S_n)$, $S_p \in 2^{\mathcal{F}}$ and $S_n \in \mathbb{R}^{|\mathcal{V}|}$, yields a successor state $S' = (S'_p, S'_n) \in 2^{\mathcal{F}} \times \mathbb{R}^{|\mathcal{V}|}$ as follows.*

We say that a vector $S_n = (S_1, \ldots, S_{|\mathcal{V}|})$ of numerical variables satisfies *a numerical constraint $c = (h_c, \otimes, t_c) \in \gamma$ if $s_{\phi(h_c)} \otimes eval(S_n, t_c)$ is true, where $eval(S_n, t_c) \in \mathbb{R}$ is obtained by substituting all $v \in \mathcal{V}$ in t_c by $S_{\phi(h_c)}$ followed by a simplification of t_c.*

If $\alpha \subseteq S_p$ and S_n satisfies *all $c \in \gamma$ then $S'_p = S_p \cup \beta_a \setminus \beta_d$ and the vector S_n is* updated *for all $m \in \delta$. We say that the vector $S_n = (S_1, \ldots, S_{|\mathcal{V}|})$ is* updated *to vector $S'_n = (S'_1, \ldots, S'_{|\mathcal{V}|})$ by modifier $m = (h_m, \oplus, t_m) \in \delta$, if*

- $S'_{\phi(h_m)} = eval(S_n, t_m)$ *for* $\oplus = \leftarrow$,
- $S'_{\phi(h_m)} = S_{\phi(h_m)} + eval(S_n, t_m)$ *for* $\oplus = \uparrow$, *and*
- $S'_{\phi(h_m)} = S_{\phi(h_m)} - eval(S_n, t_m)$ *for* $\oplus = \downarrow$.

The propositional update $S'_p = S_p \cup \beta_a \setminus \beta_d$ is defined as in standard STRIPS. The set of goal states $\mathcal{G}$ is often given as $\mathcal{G} = (\mathcal{G}_p, \mathcal{G}_n)$ with a partial propositional state descrition $\mathcal{G}_p \subset \mathcal{F}$, and G_n as a set of numererical preconditions $c = (h_c, \otimes, t_c)$. Moreover, the arithmetic trees t_c usually collaps to simple leaves labeled with numerical constants. Hence, we might assume that $|\mathcal{G}_n| \leq |\mathcal{V}|$.

3.3 Static Analysis

The static analyzer takes the domain and problem instance as an input, grounds its propositional state information and infers different forms of planner independent static information.

- Parsing: Our simple Lisp parser generates a tree of Lisp entities. It reads the input files and recognizes the domain and problem name. 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.
- Indexing: Based on the number of counted objects, first indices for the grounded predicates, functions and actions are devised. 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 function `distance` consumes 64 indices, while `fuel`, `slow-speed`, `fast-speed`, `slow-burn`, `fast-burn`, `capacity`, and `refuel-rate` each reserve eight index positions. For the quantities `total-fuel-used`, `boarding-time`, `debarking-time` only a single fact identifier is needed. Last but not least we interpret duration as an additional quantity `total-time`.
- Flattening Temporal Identifiers: 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. We will discuss the rationale of this step in Section 5.

- Grounding Propositions: *Fact-space exploration* is a relaxed enumeration of the planning problem to determine a superset of all reachable facts. Algorithmically, a FIFO fact queue is comprised. Successively extracted facts 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 enqueued. This allows to distinguish constant from fluent facts, since only the latter are reached by exploration.

- Grouping Atoms: 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 (Edelkamp & Helmert, 1999). More formally, let $\#p_i(o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n)$ be the number of objects o_i for which the fact $(p\ o_1\ \ldots\ o_n)$ is true. We establish a single-valued invariance at i if $\#p_i(o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n) = 1$. All fix object o_j, $j \neq i$, are representative of the invariance and label the group. To allow for a better encoding, some predicates like `at` and `in` are merged. In the example 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.

- Grounding Actions: 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.

- Grounding Functions: 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 `(increase (total-time) 30)`, while `zoom plane city-a city-b` has the following numerical effects: `(increase (total-time) 150)`, `(increase (total-fuel-used) 300))`, and `(decrease (fuel plane) 300)`. Refueling, however, does not reduce to a single rational number, e.g. the effects in `refuel plane city-a` only simplify to `(increase (total-time) (/ (- (750 (fuel plane)) / 12.5)))` and `(assign (fuel plane) 750)`. To evaluate the former assignment variable `total-time` has to be instantiated *on-the-fly*. This is due to the fact that the value of the quantity `fuel plane` is not constant and itself changes over time.

4 Architecture of MIPS

Figure 4 depicts the main components of MIPS and the data flow from the input definition of the domain and the problem instance to the resulting temporal plan in the output.

The planning process can be coarsely grouped into two stages, static analysis and (heuristic search) planning.

The intermediate textual format of the static analyzer in annotated grounded PDDL-like representation serves as an interface e.g. for other planners or model checkers and as an additional resource for plan visualization. Figures 5 and 6 depict an example output for the intermediate representation in the *Zeno-Travel* example.

The object-oriented framework design of MIPS allows different heuristic estimates to be combined with different search strategies, access data structures, and scheduling options.

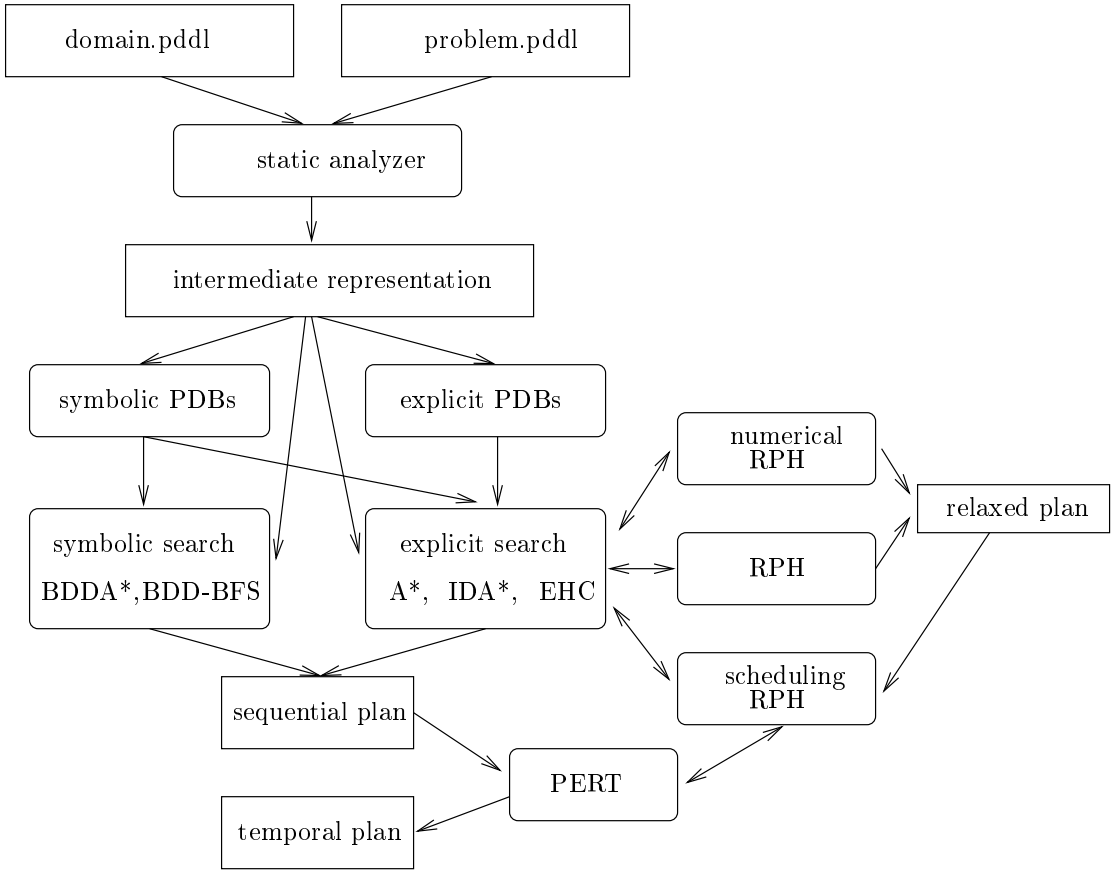

Figure 4: Architecture of MIPS

4.1 Heuristics

MIPS incorporates more than six different estimates.

- Relaxed planning heuristic (RPH): Approximation of the number of planning steps needed to solve the propositional planning problem with all delete effects removed (Hoffmann & Nebel, 2001). The heuristic is constructive, i.e. it returns the set of operators that appear in the relaxed plan.

- Numerical relaxed planning heuristic (numerical RPH): Our numerical extension to RPH is a combined propositional and numerical forward and backward approximation scheme, also allowing for multiple operator application. Our version for integrating numbers into the relaxed planning heuristic is sound, but not as general as Hoffmann's contribution (Hoffmann, 2002a): it restricts to variable-to-constant comparisons, and lacks the simplification of linear constraints.

- Pattern databases heuristic (explicit PDB heuristic): Explicit PDBs were already mentioned in the historical overview of MIPS. The different abstractions are found in a greedy best-fit bin-packing manner, yielding a selection of large PDBs in form of perfect hash tables that fit into main memory. If necessary, PDBs can be designed to be disjoint yielding an admissible estimate (Edelkamp, 2001c).

- Symbolic pattern database heuristic (symbolic PDB heuristic): Symbolic PDBs apply to both explicit and symbolic heuristic search engines. Due to the succinct BDD-representation of sets of states the averaged heuristic estimate can be increased while decreasing the number of nodes to be explored in the overall search. Symbolic PDBs are often orders of

```
(define (grounded zeno-travel-zeno-travel-1)
  (:fluents
     (at dan city-a)   (at dan city-b)   (at dan city-c)   (at dan city-d)
     (at ernie city-a) (at ernie city-b) (at ernie city-c) (at ernie city-d)
     (at plane city-a) (at plane city-b) (at plane city-c) (at plane city-d)
     (at scott city-a) (at scott city-b) (at scott city-c) (at scott city-d)
     (in dan plane)    (in ernie plane)  (in scott plane))
  (:variables (fuel plane) (total-fuel-used) (total-time))
  (:init
     (at dan city-c)   (at ernie city-c) (at plane city-a) (at scott city-a)
     (= (fuel plane) 750) (= (total-fuel-used) 0) (= (total-time) 0))
  (:goal  (at dan city-a)   (at ernie city-d) (at scott city-d))
  (:metric minimize (total-time) )
  (:group dan
    (at dan city-a)    (at dan city-b)   (at dan city-c)    (at dan city-d)
    (in dan plane))
  (:group ernie
    (at ernie city-a)  (at ernie city-b) (at ernie city-c)  (at ernie city-d)
    (in ernie plane))
  (:group plane
    (at plane city-a)  (at plane city-b) (at plane city-c)  (at plane city-d))
  (:group scott
    (at scott city-a)  (at scott city-b) (at scott city-c)  (at scott city-d)
    (in scott plane))
```

Figure 5: Grounded Representation of *Zeno-Travel* Domain.

magnitudes larger than explicit ones. Due to state conversion into a Boolean representation the retrieval of a heuristic estimate is slower than hashing, but still linear in the state description length (Edelkamp, 2002b),

- Scheduling relaxed plan heuristic (scheduling RPH, SRPH): Critical-path analysis by PERT scheduling may also guide the plan finding phase. Different to the RPH heuristics, which computes the length of the greedily extracted plan, SRPH also takes the sequence of operators into account and searches for a good parallel arrangement. Adding PERT-schedules for the path to a state and for the sequence of actions in the relaxed plan is not as accurate as the PERT-schedule of the combined paths. Therefore, the classical merit function of A*-like search engines $f = g + h$ of generating path length g and heuristic estimate h is not immediate. We define the heuristic value of SRPH as the parallel plan length of the combined path minus the parallel plan length of the generating path.

- One suitable combination of the PDB heuristic and RPH heuristics that is also implemented in MIPS, compares the retrieved result of the PDBs with the set of operators in the plan graph that respect the abstraction. The intuition is to slice the relaxed plan graph. If in the backward exploration an add-effect is selected the match will be assigned to its fact group. If the number of matches in an abstraction is smaller than the retrieved PDB value it will be increased by the lacking amount.

In the competition, except for numerical domains we chose pure RPH for sequential plan generation and scheduling PRH for temporal domains. Only in pure numerical problems we used numerical RPH. We have experimented with (symbolic) PDBs with mixed results. Since in our implementation PDBs are purely propositional and do not allow the retrieval of the corresponding operator sets of the optimal abstract plan, we have not included PDB search in the competition version of MIPS.

```
  (:action board dan plane city-a
   :condition
     (and (at dan city-a) (at plane city-a))
   :effect
     (and (in dan plane)  (not (at dan city-a))
        (increase (total-time) (30.000000))))
  [...]
  (:action zoom plane city-a city-b
   :condition
     (and
        (at plane city-a)
        (>= (fuel plane) (300.000000)))
   :effect
     (and (at plane city-b)  (not (at plane city-a))
        (increase (total-time) (60.000000))
        (increase (total-fuel-used) (300.000000))
        (decrease (fuel plane) (300.000000))))
  [...]
  (:action refuel plane city-a
   :condition
     (and
        (at plane city-a)
        (< (fuel plane) (750.000000)))
   :effect
     (and
        (increase (total-time) (/ (- (750.000000) (fuel plane)) (12.500000)))
        (assign (fuel plane) (750.000000))))
 [...]
)
```

Figure 6: Grounded Representation of *Zeno-Travel* Domain (cont.).

4.2 Exploration Algorithms

The algorithm portfolio includes:

- Weighted A* (weighted A*/A*): The A* algorithm (Hart, Nilsson, & Raphael, 1968) can be casted as a derivate of Dijkstra's SSSP exploration on a re-weighted graph. For lower bound heuristics, original A* can be shown to generate optimal plans (Pearl, 1985). Weightening the influence of the heuristic estimate may accelerate solution finding, but also affects optimality (Pohl, 1977). The set of horizon nodes are maintained in a priority queue *Open*, while the settled nodes are kept in *Closed*.

  In MIPS, Weighted A* is implemented with a Dial or a Weak-Heap priority queue data structure (Dial, 1969; Edelkamp & Stiegeler, 2002). The former is used for propositional planning only, while the latter applies to general planning with scheduling estimates. Arrays have been implemented as a dynamic table that double their sizes if they become filled. MIPS stores all generated and expanded states in a hash table. An alternative, yet not implemented, but more flexible storage structure is collection of persistent trees as in the TL planning system (Bacchus & Kabanza, 2000), one for each predicate. In the best case queries and update times to the structure are logarithmic in the number of represented atoms.

- Weighted Iterative-Deepening A* ((W)IDA*): The memory-limited variant of (Weighted) A* is well-suited to large exploration problems with efficient evaluation functions of small integer range (Korf, 1985). In MIPS, IDA* is extended with bit-state hashing (Edelkamp

& Meyer, 2001) to improve duplicate detection with respect to ordinary transposition tables (Reinefeld & Marsland, 1994). This form of partial search effectively trades state-space coverage for completeness. For a further compression of the planning state space, all variables that appear in the objective function are neglected from hash address calculations and state comparisons.

- Enforced Hill Climbing (EHC): The approach is another compromise between exploration and exploitation. EHC searches with an improved evaluation in breadth-first manner and commits established decisions as final (Hoffmann, 2000). EHC is complete in undirected problem graphs and seems to have a slight advantage to Weighted A* when combined with RPH and other pruning cuts. On the other hand, it can be misguided in unstructured planning domains and is likely to get lost in problem graphs with dead-ends.

- Bidirectional Symbolic Breadth-First-Search (BDD-BFS): The implementation performs bidirectional blind symbolic search, choosing the next search direction in favor to the faster executions of the previous iterations (Edelkamp & Helmert, 1999).

- Weighted Symbolic A* (BDDA*): The algorithm performs guided symbolic search and takes a (possibly partitioned) symbolic representation of the heuristic as an additional input. Given a consistent estimate for a uniformly weighted graph, BDDA* performs at most $\mathcal{O}(f^{*^2})$ iterations, where f^* is the optimal solution length, where consistent estimates keep the accumulated f-values on each exploration path monotonical increasing.

- Weak and Strong Planning: These two symbolic exploration algorithms suited to non-deterministic planning have been added to MIPS (Cimatti, Roveri, & Traverso, 1998), but due to the lack of an agreed standard for a domain description language, the implementation was only tested on deterministic samples in which the above symbolic algorithms clearly perform better. The encoding scheme directly transfers to the non-deterministic scenario, where plans were stored in a form of state-action tables.

In the competition we applied Weighted A* with weight 2, e.g. the merit for all states $S \in \mathcal{S}$ was fixed as $f(S) = g(S) + 2 \cdot h(S)$, yielding good but not necessarily optimal plans. In temporal domains we introduced an additional parameter δ to scale the influence between propositional estimates ($f_p(S) = g_p(S) + 2 \cdot h_p(S)$) and scheduled ones ($f_s(S) = g_s(S) + 2 \cdot h_s(S)$). More precisely, we altered the comparison function for the priority queue, so that a comparison of parallel length priorities was invoked if the propositional difference of values was not larger than $\delta \in \mathbb{N}_0$. A higher value of δ refers to a higher influence of the SRPH, while $\delta = 0$ indicates no scheduling at all. In the competition we produced data with $\delta = 0$ (Pure MIPS), and $\delta = 2$ (optimized MIPS).

5 Temporal Planning

PDDL 2.1 domain descriptions include temporal modifiers *at start*, *over all*, and *at end*, where label *at start* denotes the preconditions and effects at invokation time of the action, *over all* refers to an invariance condition and *at end* to the finalization conditions and consequences of the action.

5.1 Temporal Model

In Figure 7 we show two different options to flatten this information back to planning with preconditions and effects to derive its semantic.

In the first case (top right), the compound operator is split into three smaller parts, one for action invocation, one for invariance maintenance, and one for action termination. This is the semantic suggested by (Fox & Long, 2001).

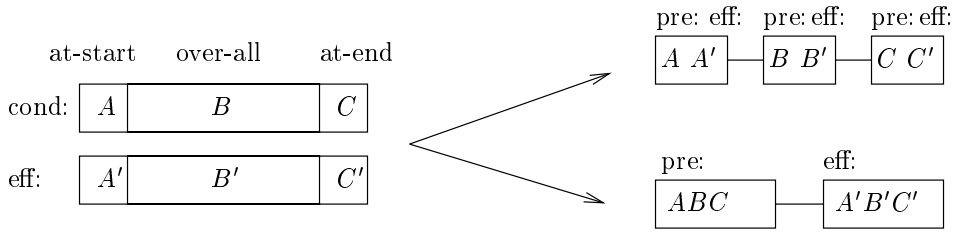

Figure 7: Compiling Temporal Modifiers into Operators.

As expected there are no effects in the invariance pattern, i.e. $B' = \emptyset$. Moreover, we found that in the benchmarks it is uncommon that new effects in `at-start` are preconditioned for termination control or invariance maintenance, i.e. $A' \cap (B \cup C) = \emptyset$.

Therefore, in MIPS the simpler second operator representation model was chosen (bottom right). The intermediate format of the example problem in Figures 5 and 6 implicitly assumed this simpler temporal model. At least for sequential plan finding we have not observed any deficiencies by assuming this temporal model, in which each action starts immediately after a previous one has terminated.

This simple temporal model motivates the definition of the first plan objective: the sequential plan.

Definition 4 *(Sequential Plan) A* sequential plan $\pi_s = (O_1, \ldots, O_k)$ *is an ordered sequence of operators* $O_i \in \mathcal{O}$, $i \in \{1, \ldots, k\}$, *that transforms the initial state* $\mathcal{I}$ *into one of the goal states* $G \in \mathcal{G}$, *i.e., there exists a sequence of states* $S_i \in \mathcal{S}$, $i \in \{0, \ldots, k\}$, *with* $S_0 = \mathcal{I}$, $S_k = G$ *and* S_i *is the outcome of applying* O_i *to* S_{i-1}, $i \in \{1, \ldots, k\}$.

Minimizing sequential plan length was the only objective in the first and second planning competition. Since *Graphplan*-like planners (Blum & Furst, 1995) like IPP (Koehler, Nebel, & Dimopoulos, 1997) and STAN (Long & Fox, 1998) already produced parallel plans, this was a indeed a limiting aspect to evaluate plan quality. The most important reason for this artificial restriction was that total-ordered plans were easier accessible for automated validation, a necessity for evaluating correctness in a competitive scenario.

5.2 Operator Dependency

The formal definition of operator dependency allows to avoid the transpositioned generation of independent actions and, more importantly, enables optimal schedules of sequential plans with respect to the generated action sequence and its causal structure. If all operators are dependent (or void with respect to the optimizer function), the problem is inherent sequential and no schedule leads to any improvement.

Definition 5 *(Dependency/Mutex Relation) Two grounded operators* $o = (\alpha, \beta, \gamma, \delta)$ *and* $o' = (\alpha', \beta', \gamma', \delta')$ *in* $\mathcal{O}$ *are* dependent/mutex, *if one of the following three conditions holds:*

1. *The propositional precondition set of one operator has a non-empty intersection with the add or the delete lists of the other one, i.e.,* $\alpha \cap (\beta'_a \cup \beta'_d) \neq \emptyset$ *or* $(\beta_a \cup \beta_d) \cap \alpha' \neq \emptyset$.

2. *The head of a numerical modifier of one operator is contained in some condition of the other one, i.e. there exists a* $c' = (h'_c, \otimes, t'_c) \in \gamma'$ *and a* $m = (h_m, \oplus, t_m) \in \delta$ *with* $h_m \in \text{LeafVariables}(t'_c) \cup \{h'_c\}$ *or there exists a* $c = (h_c, \otimes, t_c) \in \gamma$ *and a* $m' = (h'_m, \oplus, t'_m) \in \delta'$ *with* $h'_m \in \text{LeafVariables}(t_c) \cup \{h_c\}$. *Intuitively, an operator modifies variables that appear in the condition of the other. This may be referred to as a* direct conflict.

3. *The head of the numerical modifier of one operator is contained in the formula body of the modifier of the other one, i.e., there exists a* $m = (h_m, \oplus, t_m) \in \delta$ *and* $m' = (h'_m, \oplus, t'_m) \in \delta'$ *with* $h_m \in \text{LeafVariables}(t'_m)$ *or* $h'_m \in \text{LeafVariables}(t_m)$. *This may be referred to as an* indirect conflict.

The dependence relation may be refined according to the PDDL 2.1 guidelines for mutual exclusion (Fox & Long, 2001), but for our purposes for improving sequential plans this approach is sufficient. In our implementation (at least for temporal and numerical planning) the dependence relation is computed beforehand and tabulated for constant time access. To improve the efficiency of pre-computation, the set of leaf variables is maintained in an array, once the grounded operator is constructed.

To detect domains for which any parallelization leads to no improvement, a planning domain is said to be *inherently sequential* if all operators in any sequential plan are dependent or instantaneous (i.e. with zero duration). The static analyzer checks this by testing each operator pair. While some benchmark domains like *DesertRats* and *Jugs-and-Water* are inherently sequential, others like *ZenoTravel* and *Taxi* are not.

Operator independence also indicates transpositions of two operators o_1 and o_2 to safely prune exploration in sequential plan generation.

Definition 6 *(Parallel Plan) A* parallel plan $\pi_c = ((O_1, t_1), \ldots, (O_k, t_k))$ *is a* schedule *of operators* $O_i \in \mathcal{O}$, $i \in \{1, \ldots, k\}$, *that transforms the initial state* $\mathcal{I}$ *into one of the goal states* $G \in \mathcal{G}$, *where* O_i *is executed at time* t_i.

(Bäckström, 1998) clearly distincts partial ordered plans $(O_1, \ldots, O_k, \preceq)$, with the relation $\preceq \subseteq \{O_1, \ldots, O_k\}^2$ being a partial order (reflexive, transitive, and antisymmetric), from parallel plans $(O_1, \ldots, O_k, \preceq, \#)$, with $\# \subseteq (\preceq \cup \preceq^{-1})$ (irreflexive, symmetric) expressing, which actions must not be executed in parallel.

Definition 7 *(Precedence Ordering) A ordering* $\preceq_d$ *induced by the set of operators* $\{O_1, \ldots, O_k\}$ *and a dependency relation is given by* $O_i \preceq_d O_j$, *if* O_i *and* O_j *are dependent and* $1 \leq i < j \leq k$.

Precedence is not a partial ordering, since it is neither reflexive nor transitive. By computing the transitive closure of the relation, however, precedence could be extended to a partial ordering. A sequential plan $O_1, \ldots, O_k$ produces an acyclic set of precedence constraints $O_i \preceq_d O_j$, $1 \leq i < j \leq k$, on the set of operators. It is also important to observe, that the constraints are already topologically sorted according to $\preceq_d$ by taking the node ordering $\{1, \ldots, k\}$.

Definition 8 *(Respecting Precedence Ordering in Parallel Plan) Let* $d(O)$ *for* $O \in \mathcal{O}$ *be the* duration *of operator* O *in a sequential plan. For a parallel plan* $\pi_c = ((O_1, t_1), \ldots, (O_k, t_k))$ *that respect* $\preceq_d$, *we have* $t_i + d(O_i) \leq t_j$ *for* $O_i \preceq_d O_j$, $1 \leq i < j \leq k$.

For optimizing plans (Bäckström, 1998) defines *parallel execution time* as $\max\{t_i + d(O_i) \mid O_i \in \{O_1, \ldots, O_k\}\}$, so that if $O_i \preceq O_j$, then $t_i + d(O_i) \leq t_j$, and if $O_i \# O_j$, then either $t_i + d(O_i) \leq t_j$ or $t_j + d(O_j) \leq t_i$. These two possible choices in # are actually not apparent in practice, since we already have a precedence relation at hand and just seek the optimal arrangement of operators. In contrast we assert that only one option, namely $t_i + d(O_i) \leq t_j$ can be true, reducing # to $\preceq_d$. More importantly, (Bäckström, 1998)'s work introduces unnecessary time complexity, since optimized scheduling a set of fixed-timed operators is already an NP-complete problem.

Definition 9 *(Optimal Parallel Plan) An optimal parallel plan with respect to a sequence of operators* $O_1, \ldots, O_k$ *and precedence ordering* $\preceq_d$ *is a parallel plan* $\pi^* = ((O_1, t_1), \ldots, (O_k, t_k))$ *with minimal parallel execution time* $OPT = \max\{t_i + d(O_i) \mid O_i \in \{O_1, \ldots, O_k\}\}$ *among all parallel plans* $\pi_c = ((O_1, t'_1), \ldots, (O_k, t'_k))$ *that respect* $\preceq_d$.

Procedure *Critical-Path*
Input: *Sequence of operators* $O_1, \ldots, O_k$, precedence ordering $\preceq_d$
Output: Optimal parallel plan length $\max\{t_i + d(O_i) \mid O_i \in \{O_1, \ldots, O_k\}\}$
for all $i \in \{1, \ldots, k\}$
$e(O_i) = d(O_i)$
for all $j \in \{1, \ldots, i-1\}$
if $(O_j \preceq_d O_i)$
if $e(O_i) < e(O_j) + d(O_i)$
$e(O_i) \leftarrow e(O_j) + d(O_i)$
return $\max_{1 \leq i \leq k} e(O_i)$

Table 2: Algorithm to Compute Critical Path Length.

Many algorithms have been suggested to convert sequential plans into partial ordered ones (Pednault, 1986; Regnier & Fade, 1991; Veloso, Pérez, & Carbonell, 1990). Most of them interpret a total ordered plan as a maximal constrained partial ordering $\preceq = \{(O_1, O_j) \mid 1 \leq i < j \leq k\}$ and search for least constraint plans. However, the problem of minimum constraint "deordering" has also been proven to be NP-hard, except if the so-called validity check is polynomial (Bäckström, 1998), where deordering maintains validity of the plan by lessening its constraintness, i.e. $\preceq' \subseteq \preceq$ for a new ordering $\preceq'$.

Since we have an explicit model of dependency and time, optimal parallel plans will not change the ordering relation $\preceq_d$ at all.

5.3 Critical Path Analysis

The *Project Evaluation and Review Technique* (PERT) is a critical path analysis algorithm usually applied to project management problems. The critical path is established, when the total time for activities on this path is greater than any other path of operators. A delay in any tasks on the critical path leads to a delay in the project. The heart of PERT is a network of tasks needed to complete a project, showing the order in which the tasks need to be completed and their dependencies between them. As shown in Table 2, PERT scheduling reduces to a derivate of Dijkstra's single shortest path algorithm within acyclic graphs (Cormen, Leiserson, & Rivest, 1990).

In the algorithm, $e(O_i)$ is the tentative earliest end time of operator O_i, $i \in \{1, \ldots, k\}$, while the earliest starting times t_i for all operators in the optimal plan are given by $t_i = e(O_i) - d(O_i)$.

Theorem 1 *(PERT Scheduling) Given a sequence of operators* $O_1, \ldots, O_k$ *and a precedence ordering* $\preceq_d$ *an optimal parallel plan* $\pi^* = ((O_1, t_1), \ldots, (O_k, t_k))$ *can be computed in optimal time* $\mathcal{O}(k + \mid \preceq_d \mid)$.

Proof: The proof is done by induction on $i \in \{1, \ldots, k\}$. The induction hypothesis is that after iteration i the value $e(O_i)$ is correct, e.g. $e(O_i)$ is the earliest end time of operator O_i. This is clearly true for $i = 1$, since $e(O_1) = d(O_1)$. We now assume that the hypothesis is true $1 \leq j < i$ and look at iteration i. There are two choices. Either there is a $j \in \{1, \ldots, i-1\}$ with $(O_j \preceq_d O_i)$. For this case after the inner loop is completed, $e(O_i)$ is set to $\min\{e(O_j) + d(O_j) \mid O_j \preceq_d O_i, j \in \{1, \ldots, i-1\}\}$. On the other hand, $e(O_i)$ is optimal, since O_i cannot start earlier than $\min\{e(O_j) \mid O_j \preceq_d O_i, j \in \{1, \ldots, i-1\}\}$, since all values $e(O_j)$ are already the smallest possible by induction hypothesis. If there is no $j \in \{1, \ldots, i-1\}$ with

$(O_j \preceq_d O_i)$, then $e(O_i) = d(O_i)$ as in the base case. Therefore, at the end $\max_{1 \leq i \leq k} e(O_i)$ is the optimal parallel path length.

The time and space complexities of the algorithm *Critical-Path* are clearly in $\mathcal{O}(k^2)$, where k is the length of the sequential plan. Using an adjacency list representation these efforts can be reduced to time and space proportional to the number of vertices and edges in the dependence graph, which are of size $\mathcal{O}(k + | \preceq_d |)$. The bound is optimal, since the input consists of $\Theta(k)$ operators and $\Theta(| \preceq_d |)$ dependencies among them. □

5.4 Graphplan Distances

In this section we restrict the planning model to *valid STRIPS plans* as in the original article of *Graphplan* (Blum & Furst, 1995), where the execution cost of each operator is 1 and the semantics of a parallel plan are as follows.

For each time step i, $i \in \{1, \ldots, l\}$, a state $S_i \in \mathcal{S}$ is generated by applying all operators with time stamp $i - 1$ to S_{i-1}, where $S_0 = \mathcal{I}$. An optimal parallel plan is a parallel plan of minimal length l. The name *dependency* is borrowed from the notion of partial order reduction in explicit-state model checking (Clarke et al., 1999), where two operators O_1 and O_2 are *independent* if for each state $S \in \mathcal{S}$ the following two properties hold:

1. *Enabledness* is preserved, i.e. O_1 and O_2 do not disable each other.
2. O_1 and O_2 are *commutative*, i.e. executed in any order O_1 and O_2 lead to the same state.

Two actions *interfere*, if they are dependent. The original *Graphplan* definition is very closed to ours, which fixes interference as $\beta'_d \cap (\beta_a \cup \alpha) \neq \emptyset$ and $(\beta'_a \cup \alpha') \cap \beta_d \neq \emptyset$.

Lemma 1 *If $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$, operator inference in the* Graphplan *model is implied by the propositional MIPS model of dependence.*

Proof: If $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$, for two independent operators $o = (\alpha, \beta)$ and $o' = (\alpha', \beta')$: $\alpha \cap (\beta'_a \cup \beta'_d) = \emptyset$ implies $\beta_d \cap (\beta'_a \cup \beta'_d) = \emptyset$, which in turn yields $\beta_a \cap \beta'_d = \emptyset$. The condition $\beta'_a \cap \beta_d = \emptyset$ can be inferred analogously by exchanging primed and unprimed variables. □

Theorem 2 *Two independent STRIPS operators $o = (\alpha, \beta)$ and $o' = (\alpha', \beta')$ in $\mathcal{O}$ with $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$ are* enabledness preserving *and* commutative, *i.e. for all states in $S \subseteq 2^{|A|}$ we have $o(o'(S)) = o'(o(S))$.*

Proof: Since $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$, we have $\beta_a \cap \beta'_d = \emptyset$ and $\beta'_a \cap \beta_d = \emptyset$ by Lemma 1. Let S' be the state $((S \setminus \beta_d) \cup \beta_a)$ and let S'' be the state $((S \setminus \beta'_d) \cup \beta'_a)$. Since $(\beta'_a \cup \beta'_b) \cap \alpha = \emptyset$, o is enabled in S'', and since $(\beta_a \cup \beta_b) \cap \alpha' = \emptyset$, o' is enabled in S'. Moreover,

$$
\begin{aligned}
o(o'(S)) &= (((S \setminus \beta'_d) \cup \beta'_a) \setminus \beta_d) \cup \beta_a \\
&= (((S \setminus \beta'_d) \setminus \beta_d) \cup \beta'_a) \cup \beta_a \\
&= S \setminus (\beta'_d \cup \beta_d) \cup (\beta'_a \cup \beta_a) \\
&= S \setminus (\beta_d \cup \beta'_d) \cup (\beta_a \cup \beta'_a) \\
&= (((S \setminus \beta_d) \setminus \beta'_d) \cup \beta_a) \cup \beta'_a \\
&= (((S \setminus \beta_d) \cup \beta_a) \setminus \beta'_d) \cup \beta'_a = o'(o(S))
\end{aligned}
$$

□

A less restrictive notion of independence, in which several actions may occur at the same time even if one deletes an add-effect of another is provided in (Knoblock, 1994).

All three models of valid plans are restrictive, since they assume that for each parallel plan there exist at least one corresponding total ordered plan. In general, however, this is not true. Consider the simple STRIPS planning problem domain with $\mathcal{I} = \{B\}$, $\mathcal{G} = \{\{A, C\}\}$, and $\mathcal{O} = \{(\{B\}, \{A\}, \{B\}), (\{B\}, \{C\}, \{B\})\}$. Obviously, both operators are needed for goal achievement, but there is no sequential plan of length 2, since B is deleted in both operators. However, a parallel plan could be executed, since all precondition are fulfilled at the first time step.

5.5 Full Enumeration Algorithms

Even though full state-space enumaration is far from being practical they provide a basis for heurisitic search engines. In optimal parallel plans, each operator either starts or ends at the start or end time of another operator. Therefore, for a fixed number of operators, we can assume a possibly exponential but finite number of possible parallel plans.

This immediately leads to the following plan enumeration algorithm *ENUM-1*. For all $|\mathcal{O}|^i$ operator sequences of length i, $i \in \mathbb{N}$, generate all possible partial orderings, check for each individual schedule if it transforms the initial state into one of the goals, and take the sequence with smallest parallel plan length. Since all parallelizations are computed we have established the following result.

Theorem 3 *If the number of operators for an optimal parallel plan is bounded, ENUM-1 is complete and computes optimal parallel plans.*

Note that the first i with a matching solution does not necessarily yield an optimal parallel path, since longer operator sequences might rise better parallel solutions. ENUM-1 can also generate non-valid plans in the *Graphplan* model. For a better distinction between the objectives for parallel plans, we keep the notion of *validity* in this section.

Assuming only valid plans implies that each parallel plan corresponds to at least one (possible many) sequential ones. Viewed from the opposite side, each partial-ordered plan can be established by generating a totally-ordered plan first and then apply a scheduling algorithm to it to find its best partial-order. Therefore, the next two enumeration schemes produce valid plans only.

Enumeration algorithm *ENUM-2* generates all feasible sequential plans of length i with increasing $i \in \mathbb{N}$, and computes their optimal schedule with respect to the number of operators and dependency property. Since optimal parallelization of all valid operator sequences are computed we have established the following theorem.

Theorem 4 *If the number of operators for an optimal parallel plan is bounded, ENUM-2 is complete and computes a valid optimal parallel plan.*

A complete enumeration scheme of all sequential plans that transform the initial state into one goal state is also still computationally expensive, but ruling out impossible operator applications drastically reduces the vast number of $|\mathcal{O}|^i$ operator sequences of length i. Bäckström's result for deriving partial orders has shown, that given the sequence of operators in a sequential plan, to infer an optimized partial order that respects a set of mutexes is NP-hard, so that even for the second phase no polynomial-time algorithm is to be expected. Therefore, at least for STRIPS we have restricted the PSPACE-hard planning task (Bylander, 1994) to an NP-hard problem for each generated sequential plan.

When the concept of mutual exclusion is extended to a precedence relation between operators, there exist at least one sequential plan that respects the set of operators and the set of precedence constraints. From the opposite point of view, for each sequential plan there exist at least one parallel plan that respects both the number of operators and the imposed set of precedence constraints.

Algorithm *ENUM-3* is a straight variant of *ENUM-2* that simply applies PERT scheduling for finding the optimal parallel plan, with the main difference that it additionally maintains the causal structure.

We have seen that *ENUM-1* may generate parallel plans that *ENUM-2* cannot produce. Are there also valid plans that *ENUM-2* can produce, but *ENUM-3* cannot? The answer is no. If *ENUM-2* terminates with an optimal schedule, we generate a corresponding sequential plan while preserving the causal structure. Optimal PERT-scheduling of this plan with respect to the set of operators and the imposed precedence relation will yield back the optimal parallel plan. Since all sequential paths are eventually generated, the given partial will also be found by *ENUM-3*. This proves the following result.

Theorem 5 *If the number of operators for an optimal parallel plan is bounded, ENUM-3 is complete and computes a valid optimal parallel plan.*

In the following, we interpret optimized parallel plans as nodes in a weighted directed graph $G = (V, E, w)$. Edges correspond to possible extensions of the plans with an additional operator, which can be found by a sequentialization of the parallel plan followed by a PERT scheduling operation. The weight function denotes the difference in parallel plan length. Since the set of operators and the precedence set is enlarged, all weights will be greater than or equal to 0. If only a finite number of actions can be executed in parallel, then any infinite path in G has unbounded cost. Therefore, we can traverse G in shortest path ordering using Dijkstra's algorithm to finally yield an optimal parallel plan.

The argument for optimality is that Dijkstra's algorithm is complete, i.e., it cannot exit with failure, since if the horizon list becomes becomes empty there is no solution at all. If the horizon is not empty, there is at least one node on an optimal solution path, which has to be selected before any goal node with larger cost.

Theorem 6 *If only a finite number of actions can be executed in parallel, Dijkstra's shortest path enumeration is complete and computes a valid optimal parallel plan.*

5.6 Heuristic Search Enumeration

The enumeration algorithms in the previous section are sound, complete and optimal in theory. On the other hand enumeration schemes do not contradict known undecidability results in numerical planning (Helmert, 2002). If we have no additional information like a bound to the maximal number of actions in a plan or on the number of actions that can be executed in parallel, we cannot say if the enumeration will terminate or not.

The main drawback of the above approaches is that they are seemingly too slow for practical planning. Heuristic search algorithms like A* and IDA* reorder the traversal of states in the planning problem, and an admissible estimate does not affect completeness and optimality. The reason for completeness in finite graphs is that the number of acyclic paths in G is finite and with every node expansion, A* adds new links to its traversal tree. Each newly added link represents a new acyclic path, so that the reservoir of path must eventually be exhausted. The argument is valid for any best-first strategy that prunes cyclic paths, but by their move-committing nature, hill-climbing algorithms are not complete.

(Pearl, 1985) has shown that A* is complete even on infinte graphs, demanding that the cost of every infinite path is unbounded. A deeper investigation shows that given an admissible estimate there must always be a node in the current search horizon with optimal priority. Actually to preserve this condition for admissible but not necessarily consistent estimates, already expanded node may have to be reconsidered (re-opening). Hence, A* must also terminate with an optimal solution.

Theorem 7 *If the cost of every infinite plan is unbounded, A* enumeration with an admissible parallel plan length estimate computes a valid optimal parallel plan.*

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
130: (board ernie plane city-c) [30]
160: (refuel plane city-c) [40]
200: (zoom plane city-c city-a) [100]
300: (debark dan plane city-a) [20]
320: (board scott plane city-a) [30]
350: (refuel plane city-a) [40]
390: (zoom plane city-a city-c) [100]
490: (refuel plane city-c) [40]
530: (zoom plane city-c city-d) [100]
630: (debark ernie plane city-d) [20]
650: (debark scott plane city-d) [20]
```

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
100: (board ernie plane city-c) [30]
100: (refuel plane city-c) [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a) [20]
240: (board scott plane city-a) [30]
240: (refuel plane city-a) [40]
280: (zoom plane city-a city-c) [100]
380: (refuel plane city-c) [40]
420: (zoom plane city-c city-d) [100]
520: (debark ernie plane city-d) [20]
520: (debark scott plane city-d) [20]
```

Figure 8: A Sequential Plan for *Zeno-Travel* (left) and its PERT Schedule (right).

Note that the assumption of unbounded sequential plan costs is not true in all benchmark problems, since there may be an infinite sequence of instantaneous events that do not contribute to the plan objective. For example, loading and unloading tanks in *DesertRats* does not affect `total-fuel` consumption, which is to be minimized in one benchmark instance.

As a matter of fact, informative admissible parallel plan length estimates are not easy to obtain. This was the reason in MIPS to chose sequential plan generation first, because very effective heuristics are known to generate sequential plans quickly. With the SRPH we choose a parallel plan length approximation, but since it extends PRH, it is known to be not admissible.

5.7 Pruning Anomalies

Other acceleration techniques like sequential plan hashing, symmetry and transposition cuts have to be chosen carefully to maintain parallel plan length optimality.

Take for example sequential state memorization, i.e. the memorization of states in the sequential plan generation process. This approach does affect parallel optimality, as the following example shows.

Consider the sequences

`(zoom city-a city-c plane)`, `(board dan plane)`, `(refuel plane)`,
`(zoom city-c city-a plane)`, `(board scott)`, `(debark dan)`, `(refuel plane)`,

and

`(board scott)`,`(zoom city-a city-c plane)`,`(board dan plane)`,
`(refuel plane)`, `(zoom city-c city-a plane)`, `(debark dan)`, `(refuel plane)`

in the *Zeno-Travel* problem. The set of operators is the same and so is the resulting (sequentially generated) state.

However, the PERT schedule for the first sequence is shorter than the schedule for the second one, since in the previous case the time for boarding `scott` is compensated by the remaining two operators.

For small problems, such anomalies can be avoided by avoided duplicate pruning at all. As an example Figure 8 depicts a sequential plan for the example problem instance and its PERT schedule, which turns out to be the overall optimal parallel plan.

In order to generate sequential solutions for large planning problem instances, in the competition version of MIPS we have introduced cuts that affect optimality but reduce the number of expansions significantly.

```
  0: (board scott plane city-a) [30]
 30: (fly plane city-a city-c) [150]
180: (board ernie plane city-c) [30]
180: (board dan plane city-c) [30]
210: (fly plane city-c city-a) [150]
360: (debark dan plane city-a) [20]
360: (refuel plane city-a) [53.33]
413.33: (fly plane city-a city-c) [150]
563.33: (fly plane city-c city-d) [150]
713.33: (debark ernie plane city-d) [20]
713.33: (debark scott plane city-d) [20]
```

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
100: (board ernie plane city-c) [30]
100: (refuel plane city-c) [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a) [20]
240: (board scott plane city-a) [30]
240: (refuel plane city-a) [40]
280: (fly plane city-a city-c) [150]
430: (fly plane city-c city-d) [150]
580: (debark ernie plane city-d) [20]
580: (debark scott plane city-d) [20]
```

Figure 9: Optimized Plans in *Zeno-Travel* according to different Plan Objectives.

5.8 Arbitrary Plan Objectives

In PDDL 2.1 different plan metrics can be devised. In Figure 9 we depict two plans found by MIPS when modifying the objective function from minimizing `total-time` to minimize `total--fuel-used`, and to minimize the compound `(+ (* 10 (total-time)) (* 1 (total-fuel--used)))`.

For the first case we computed an optimal value of 1,333.33, while for the second case we established 7,666.67 as the optimized merit. When optimizing time, the ordering of board and zoom actions is important. When optimizing *total-fuel* we reduce speed to save fuel consumption to 333.33 per flight but we may board the first passenger immediately. We also save two refuel actions with respect to the first case.

When increasing the importance of time we can trade refueling actions for time, so that both zooming and flight actions are chosen for the complex minimization criterion.

We first thought, that we could simply substitute the plan objective in the PERT scheduling process. However, the results did not match with the ones produced by the validator (Long & Fox, 2001a), in which the final time is substituted in the objective function after the plan has been build.

The way we evaluate objective functions that include time is as follows. First we schedule the (relaxed or final) sequential plan. Then we temporarily substitute the `total-time` value in the state with the parallel plan length and evaluate the formula to get the objective function value. To avoid conflicts in subsequent expansions, afterwards we set the value `total-time` back to the optimal one in the sequential plan.

6 Symmetry

An important feature of parameterized predicates, functions and action descriptions in the domain specification file is that actions are transparent to different bindings of parameters to objects. Disambiguating information is present in the problem instance file.

In case of typed domains, many planners, including MIPS, compile all type information into additional predicates, attach additional preconditions to actions and enrich the initial states by suitable object-to-type atoms.

As a consequence, a symmetry is viewed as a permutation of objects that is present in the current state, in the goal representation, and transparent to the set of operators.

There are $n!$, $n = |\mathcal{OBJ}|$, possible permutations of the set of objects. Taking into account all type information reduces the number of all possible permutation to

$$\binom{n}{t_1, \ldots, t_k} = \frac{n!}{t_1! \cdot \ldots \cdot t_k!}.$$

where t_i is the number of objects with type i, $i \in \{1, \ldots, k = |\mathcal{TYPES}|\}$. In a moderate sized logistic domain with 10 cities, 10 trucks, 5 airplanes, and 15 packages, this results in $40!/(10! \cdot 10! \cdot 5! \cdot 15!) \geq 10^{20}$ permutations.

To reduce the number of potential symmetries to a tractable size we restrict symmetries to object transpositions, for which we have at most $n(n-1)/2 \in \mathcal{O}(n^2)$ candidates. Including type information this number further reduces to

$$\sum_{i=1}^{k} \binom{t_i}{2} = \sum_{i=1}^{k} t_i(t_i - 1)/2.$$

In the following, the set of typed object transpositions is denoted by $\mathcal{SYMM}$. For the example, we have $|\mathcal{SYMM}| = 45 + 45 + 10 + 105 = 205$.

6.1 Static Symmetries

We generate a set of object pairs $(o, o') \in \mathcal{SYMM}$, indistinguishable with respect to the set of instantiated operators and the goal specification.

Definition 10 *(Object Transpositions for Fluents, Variables, and Operators) A transposition of objects $(o, o') \in \mathcal{SYMM}$ applied to a fluent $f = (p\ o_1, \ldots, o_{k(p)}) \in \mathcal{F}$, written as $f[o \leftrightarrow o']$, is defined as $(p\ o'_1, \ldots, o'_{k(p)})$, with $o'_i = o_i$ if $o_i \notin \{o, o'\}$, $o_i = o'$ if $o_i = o$, and $o_i = o$ if $o_i = o'$, $i \in \{1, \ldots, k(p)\}$. Object transpositions $[o \leftrightarrow o']$ applied to a variable $v = (f\ o_1, \ldots, o_{k(f)}) \in \mathcal{V}$ or to an operator $O = (a\ o_1, \ldots, o_{k(a)}) \in \mathcal{O}$ are defined analogously.*

By definition we have

Lemma 2 *For all $f \in \mathcal{F}$, $v \in \mathcal{V}$, $O \in \mathcal{O}$, and $(o, o') \in \mathcal{SYMM}$: $f[o \leftrightarrow o'] = f[o' \leftrightarrow o]$, $v[o \leftrightarrow o'] = v[o' \leftrightarrow o]$, $O[o \leftrightarrow o'] = O[o' \leftrightarrow o]$, $f[o \leftrightarrow o'][o \leftrightarrow o'] = f$, $v[o \leftrightarrow o'][o \leftrightarrow o'] = v$, and $O[o \leftrightarrow o'][o \leftrightarrow o'] = O$.*

The time complexity for checking $f[o \leftrightarrow o']$ is of order $\mathcal{O}(k(p))$. By precomputing a $\mathcal{O}(|\mathcal{SYMM}| \cdot |\mathcal{F}|)$ sized table containing the index of $f' = f[o \leftrightarrow o']$ for all $(o, o') \in \mathcal{SYMM}$, this time complexity can be reduced to $\mathcal{O}(1)$.

Definition 11 *(Object Transpositions for States) An* object transposition $[o \leftrightarrow o']$ *applied to state $S = (S_p, S_n) \in \mathcal{S}$ with $S_n = (v_1, \ldots, v_k)$, $k = |\mathcal{V}|$, written as $S[o \leftrightarrow o']$, is equal to $(S_p[o \leftrightarrow o'], S_n[o \leftrightarrow o'])$ with*

$$S_p[o \leftrightarrow o'] = \{f' \in \mathcal{F} \mid f \in S_p \ \wedge \ f' = f[o \leftrightarrow o']\}$$

and $S_n[o \leftrightarrow o'] = (v'_1, \ldots, v'_k)$ with $v_i = v'_j$ if $\phi^{-1}(i)[o \leftrightarrow o'] = \phi^{-1}(j)$ for $i, j \in \{1, \ldots, k\}$.

The time complexity to compute $S_n[o \leftrightarrow o']$ is $\mathcal{O}(k)$, since testing $\phi^{-1}(i)[o \leftrightarrow o'] = \phi^{-1}(j)$ is available in time $\mathcal{O}(1)$ by building another $\mathcal{O}(|\mathcal{SYMM}| \cdot |\mathcal{V}|)$ sized precomputed look-up table. We summarize the complexity issues as follows.

Lemma 3 *The time complexity to compute $S[o \leftrightarrow o']$ for state $S = (S_p, S_n) \in \mathcal{S}$ and $(o, o') \in \mathcal{SYMM}$ is $\mathcal{O}(|S_p| + |\mathcal{V}|)$ using $\mathcal{O}(|\mathcal{SYMM}| \cdot (|\mathcal{F}| + |\mathcal{V}|)$ space.*

Definition 12 *(Object Transpositions for Domains) A planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is symmetric with respect to the object transposition $[o \leftrightarrow o']$, abbreviated as $\mathcal{P}[o \leftrightarrow o']$, if $\mathcal{I}[o \leftrightarrow o'] = \mathcal{I}$ and for all $G \in \mathcal{G}$: $G[o \leftrightarrow o'] \in \mathcal{G}$.*

Applying Lemma 3 for all $(o, o') \in \mathcal{SYMM}$ yields

Theorem 8 *Assuming a description complexity $\mathcal{O}(|\mathcal{G}_p| + |\mathcal{V}|)$ for the set of goals $\mathcal{G}$, checking whether a planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is symmetric with respect to the object transpositions $[o \leftrightarrow o']$, with $(o, o') \in \mathcal{SYMM}$ can be done in time $\mathcal{O}(|\mathcal{SYMM}| \cdot (|\mathcal{G}_p| + |\mathcal{I}_p| + |\mathcal{V}|)$.*

Lemma 4 *If operator O is applicable in S and $S = S[o \leftrightarrow o']$ then $O[o \leftrightarrow o']$ is applicable in S and*

$$O(S)[o \leftrightarrow o'] = O[o \leftrightarrow o'](S)$$

Proof: If O is applicable in S the $O[o \leftarrow o']$ is applicable in $S[o \leftarrow o']$. Since $S = S[o \leftrightarrow o']$, $O[o \leftrightarrow o']$ applicable in S, and

$$O[o \leftrightarrow o'](S) = O[o \leftrightarrow o'](S[o \leftrightarrow o']) = O(S)[o \leftrightarrow o'].$$

□

Lemma 4 indicates how symmetry will be used to reduce exploration. If a planning problem with current state $\mathcal{C} \in \mathcal{S}$ is symmetric with respect to the operator transposition $[o \leftrightarrow o']$ then either the application of operator $O \in \mathcal{O}$ or the application of operator $O[o \leftrightarrow o']$ is neglected, significantly reducing the branching factor.

6.2 Dynamic Symmetries

One problem is that symmetries that are present in the initial state may vanish or reappear during exploration. In the *DesertRats* domain, for example, the initial set of supply tanks is indistinguishable so that only one should be loaded into the truck. Once the fuel level of the supply tanks decrease or tanks are transported to another location, formerly existing symmetries are broken. However, when two tanks in one location are emptied they can once more be considered as being symmetric.

In a forward chaining planner goal conditions do not change over time, only the initial state $\mathcal{I}$ transforms to the current state $\mathcal{C}$. Therefore, in a precompiling phase we refine the set $\mathcal{SYMM}$ to

$$\mathcal{SYMM}' := \{(o, o') \in \mathcal{SYMM} \mid \forall\, G \in \mathcal{G} : G[o \leftrightarrow o'] = G\}\,.$$

Usually $|\mathcal{SYMM}'|$ is by far smaller than $|\mathcal{SYMM}|$. For the *Zeno-Travel* instance, the symmetries left in $\mathcal{SYMM}'$ are the ones of the location of `scott` and `ernie`.

Theorem 9 *Checking whether an induced planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{G} \rangle$ with current state $\mathcal{C} = (\mathcal{C}_p, \mathcal{C}_n) \in \mathcal{S}$ is symmetric with respect to the object transpositions $[o \leftrightarrow o']$, $(o, o') \in \mathcal{SYMM}'$, can be performed in time $\mathcal{O}(|\mathcal{SYMM}'| \cdot (|\mathcal{C}_p| + |\mathcal{V}|))$.*

Therefore, we can efficiently compute set

$$\mathcal{SYMM}''(\mathcal{C}) := \{(o, o') \in \mathcal{SYMM}' \mid \mathcal{C}[o \leftrightarrow o'] = \mathcal{C}\}$$

of symmetries that are present in the current state. In the initial state of the example problem of *Zeno-Travel* $\mathcal{SYMM}''(\mathcal{C}) = \emptyset$, but once `scott` and `ernie` share the same location in a state $\mathcal{C} \in \mathcal{S}$ this object pair would be included in $\mathcal{SYMM}''(\mathcal{C})$.

By precomputing a $\mathcal{O}(|\mathcal{SYMM}| \cdot |\mathcal{O}|)$ sized table the index of operator $O' = O[o \leftrightarrow o']$ can be determined in time $\mathcal{O}(1)$ for each $(o, o') \in \mathcal{SYMM}'$.

Let $\Gamma(S)$ be the set of operators that are applicable in state $S \in \mathcal{S}$.

Definition 13 *The* pruning set $\Delta(S, \mathcal{SYMM}''(\mathcal{C})) \subset \Gamma(S)$ *is defined as the set of all operators that have a symmetric operator and that are not of minimal index, i.e.,* $\Delta(S, \mathcal{SYMM}''(\mathcal{C})) =$

$$\{O \in \Gamma(S) \mid \exists\, O' \in \Gamma(S) : \phi(O') > \phi(O) \text{ and } \exists\, (o, o') \in \mathcal{SYMM}''(\mathcal{C}) : O' = O[o \leftrightarrow o']\}\,.$$

The symmetry reduction *of $\Gamma(S, \mathcal{SYMM}''(\mathcal{C})) \subseteq \Gamma(S)$ with respect to the set $\mathcal{SYMM}''(\mathcal{C})$ is defined as $\Gamma(S, \mathcal{SYMM}''(\mathcal{C})) = \Gamma(S) \setminus \Delta(S, \mathcal{SYMM}''(\mathcal{C}))$.*

To shorten notation, in the following we write $\Gamma'(\mathcal{C})$ for $\Gamma(S, \mathcal{SYMM}''(\mathcal{C}))$ and $\Delta'(\mathcal{C})$ for $\Delta(S, \mathcal{SYMM}''(\mathcal{C}))$. Determining $\Gamma'(\mathcal{C})$ can be performed in time $\mathcal{O}(|\Gamma(S)|)$, since finding $O' = O[o \leftrightarrow o']$ and the indices $\phi(O')$ and $\phi(O)$ are all available in constant time.

Theorem 10 *Reducing the operator set $\Gamma(\mathcal{C})$ to $\Gamma(S, \mathcal{SYMM}''(\mathcal{C}))$ during the exploration of planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ preserves completeness and sequential optimality for all expanded states $\mathcal{C}$.*

Proof: Suppose that for some expanded state $\mathcal{C}$, reducing the operator set $\Gamma(\mathcal{C})$ to $\Gamma'(\mathcal{C})$ during the exploration of planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ does not preserve completeness and sequential optimality. Furthermore, let $\mathcal{C}$ be the state with this property that is maximal in the exploration order.

Then there is a sequential plan $\pi = \{O_1 \ldots, O_k\}$ in $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{G} \rangle$ with intermediate state sequence $S_0 = \mathcal{C}, \ldots, S_k \subseteq \mathcal{G}$. Obviously, $O_i \in \Gamma(S_{i-1})$, $i \in \{1, \ldots, k\}$. By the choice of $\mathcal{C}$ we have $O_1 \notin \Gamma'(S_0)$. Since $O_1 \notin \Gamma'(S_0)$ but $O_1 \in \Gamma(S_0)$ we have that $O_1 \in \Delta(S_0, \mathcal{SYMM}''(S_0))$. By the definition of the pruning set $\Delta'(S_0)$ there exists O_1', $\phi(O_1') > \phi(O_1)$ and $(o, o') \in \mathcal{SYMM}''(S_0)$ with $O_1' = O_1[o \leftrightarrow o'] \in \Gamma'(S_0)$ that is applicable in S_0. By Lemma 4 we have $O_1'(S_0) = S_1[o \leftrightarrow o']$.

Since $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{G} \rangle = \mathcal{P}[o \leftrightarrow o'] = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}[o \leftrightarrow o'] = \mathcal{C}, \mathcal{G}[o \leftrightarrow o'] = \mathcal{G} \rangle$, we have a sequential plan $O_1[o \leftrightarrow o'], \ldots, O_k[o \leftrightarrow o']$ with state sequence $S_0[o \leftrightarrow o'] = S_0, S_1[o \leftrightarrow o'], \ldots, S_k[o \leftrightarrow o'] = S_k$ that reaches the goal $\mathcal{G}$.

Sequential plan objectives are devised on parameterized predicates and functions, so that any cost function on $O_1[o \leftrightarrow o'], \ldots, O_k[o \leftrightarrow o']$ will be the same as on $O_1, \ldots, O_k$. This contradicts the assumption that reducing the operator set $\Gamma(\mathcal{C})$ to $\Gamma'(\mathcal{C})$ does not preserve completeness and optimality for all $\mathcal{C}$. □

If the plan objective is defined on instantiated predicates and objects, it can be symmetry breaking and to preserve optimality should be checked as an additional requirement similar to $\mathcal{G}$ and $\mathcal{I}$.

6.3 Symmetry Reduction in MIPS

The main purpose of the refined implementation in MIPS is to reduce the time for dynamic symmetry detection from $\mathcal{O}(|\mathcal{SYMM}'| \cdot (|\mathcal{C}_p| + |\mathcal{V}|))$ to time $\mathcal{O}(|\mathcal{C}_p| + |\overline{\mathcal{SYMM}'}| \cdot |\mathcal{V}|)$ by loosing some but not all structural properties.

The key observation is that symmetries are also present in fact groups according to their object representatives. Fact groups $G_i \subseteq \mathcal{F}$, $i \in \{1, \ldots, l\}$ implicitly define projections $\mathcal{P}|_i$ of the (propositional) planning space $\mathcal{P}$ by $\mathcal{P}|_i = \langle \mathcal{S}|_i, \mathcal{O}|_i, \mathcal{I}|_i, \mathcal{G}|_i \rangle$, with $\mathcal{S}|_i = G_i$, $\mathcal{I}|_i = \mathcal{I} \cap G_i$, $\mathcal{G}|_i = \bigcup_{G \in \mathcal{G}} G \cap G_i$, and $\mathcal{O}|_i = \{(\beta_a, \beta_b) \in \mathcal{O} \mid (\beta_a \cup \beta_b) \cap \mathcal{S}|_i \neq \emptyset\}$. By construction for all $S \in \mathcal{S}$ we have exactly one fact in each group true, such that S can be partitioned into $\{S_1, \ldots, S_l\}$, with $S_i \in \mathcal{S}|_i$, $i \in \{1, \ldots, l\}$.

Let $R_i \subseteq \mathcal{OBJ}$ be the set of object representatives for group G_i. If $S[o \leftrightarrow o'] = S$ then $\mathcal{S}|_i[o \leftrightarrow o'] = \mathcal{S}|_j$ in a group G_j with representative $R_i[o \leftrightarrow o']$. Hence, in MIPS we devise a symmetry relation $\overline{\mathcal{SYMM}}$ not on objects but on fact groups, i.e.

$$\overline{\mathcal{SYMM}} = \{(i, j) \mid 1 \leq i < j \leq l : R_i[o \leftrightarrow o'] = R_j\}.$$

Many objects, e.g. the objects of type `city` in *ZenoTravel*, were not selected as representatives for a single attribute invariance to build a group. These were neglected in MIPS, since we expect no symmetry on them. This reduces the set of objects $\mathcal{OBJ}$ that MIPS considers to a considerably smaller subset $\mathcal{OBJ}' = \bigcup_{\{1 \leq i \leq l\}} R_i$. In the example problem $|\mathcal{OBJ}| = 7$, and $|\mathcal{OBJ}'| = 4$.

It may also happen that more than one group has a representative $o \in \mathcal{OBJ}'$. However, if all fluent predicates p have arity $k(p) \leq 2$, which is frequently met in the benchmark domains, all

$|R_i|$ were equal to one for all i, so for all objects we get a finite partitioning into representatives, i.e. $\mathcal{OBJ}' = \dot{\bigcup}_{i \in \{1,\ldots,l\}} R_i$.

MIPS takes this conservative assumption and may leave other symmetries uncaught. It computes $\overline{\mathcal{SYMM}}$ by analyzing the subproblem structures $\mathcal{P}|_i$, $i \in \{1,\ldots,l\}$ instead of $\mathcal{P}$ itself. In case of an object symmetry $[R_i \leftrightarrow R_j]$ the groups G_i and G_j necessarily have to be isomorphic, and we can establish a bijective mapping $\psi : \mathcal{P}|_i \to \mathcal{P}|_j$ with subcomponents $\psi_{\mathcal{S}} : \mathcal{S}|_i \to \mathcal{S}|_j$ and $\psi_{\mathcal{O}} : \mathcal{O}|_i \to \mathcal{O}|_j$.

As above, static symmetries based on non-matching goal predicates were excluded, yielding a refinement $\overline{\mathcal{SYMM}'}$ of $\overline{\mathcal{SYMM}}$. Dynamic symmetries are detected for each expanded state S. The current state representation is mapped to the subgraphs $\mathcal{P}|_i$. The list of possibly symmetric groups $\overline{\mathcal{SYMM}'}$ is traversed to select pairs which obey the current instantiation. MIPS marks the groups with larger index as *visited*. This guarantees that operators of at least one group are executed. The complexity of this phase is bounded by $|S_p|$ and by $\overline{\mathcal{SYMM}'}$ and yields a list $\overline{\mathcal{SYMM}''}$.

Testing the propositional part $S_p = (S_1,\ldots,S_l)$ of a state S for all symmetries reduces to test, whether $\psi_{\mathcal{S}}(S_i) = S_j$ for each $(i,j) \in |\overline{\mathcal{SYMM}'}|$ and can be performed in time $\mathcal{O}(|S_p| + |\overline{\mathcal{SYMM}'}|)$. The comparison of variables $v \in \mathcal{V}$ is implemented as described in the previous section such that for the numerical part S_n we check the remaining symmetries, for total time $\mathcal{O}(|S_p| + |\overline{\mathcal{SYMM}'}| \cdot |\mathcal{V}|)$ to fix $\overline{\mathcal{SYMM}''}$ in form of visited markings.

For each expanded state S and each matching operator $O \in \Gamma(S)$ the algorithm checks, whether an applied operator is present in a visited group, in which case it is pruned. The time complexity is in $\mathcal{O}(|\Gamma(S)|)$, since operator group containment can be preprocessed and checked in constant time.

7 Visualization

For visualization of plans we extended an existing animation system for our purposes.

Vega (Hipke, 2000) is implemented as a Client-Server architecture that runs a annotated algorithm on the server side to be visualized on the client side in a Java frontend. Annotation are visualization requests that (minorly) extend the existing source code by (simple) commands like *send point*(x,y) or *send circle*(x,y,r). In the system visualization objects can be associated with hierarchical structured identifiers. The client is used both as the user front-end and the visualization engine. Thus, it allows server and algorithm selection, input of data, running and stopping algorithms, and customization of the visualization.

It can be used to *i*) manipulate scenes with hierarchically named objects — either in the view or in an object browser that displays the object tree, *ii*) view algorithm lists at the server and display algorithm information, *iii*) apply algorithms to selected data in a view, control the algorithm execution using a VCR-like panel or the execution browser, *iv*) adjust view attributes directly or using the object browser, show several algorithms simultaneously in multiple scenes and open different views for a single scene, and *v*) load and save single scenes, complete runs, and attribute lists, export scenes in xfig or gif format.

Vega allows on-line and off-line presentations. The main purpose of the server is to make algorithms accessible through TCP/IP. The server is able to receive commands from multiple clients at the same time. It allows the client to choose from the list of available algorithms, to retrieve information about the algorithm, to specify input data, to start it and to receive output data. The server maintains a list of installed algorithms. This list may be changed without the need of stopping and restarting the server.

We have extended Vega with two respects, and call it Vepa for *Visualization of Efficient Planning Algorithms* to emphasize the planning aspect. It can be run as an interactive applet available at `www.informatik.uni-freiburg.de/~mmips/visualization`.

The first program that we added is *VepaServer* which wraps plan execution and visualizes

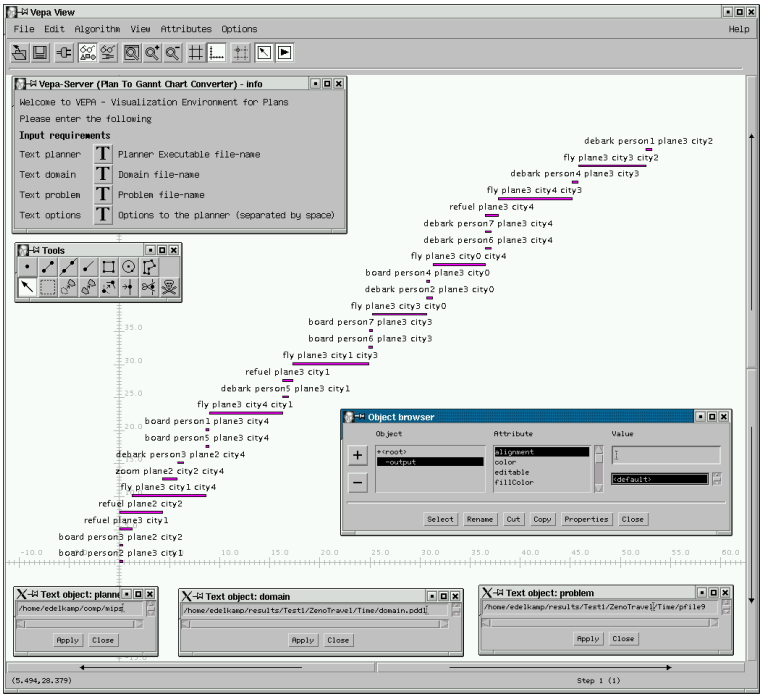

Figure 10: Visualization a Plan in Gannt Chart Format.

Gannt Charts of plans (cf. Figure 10). Gannt Charts are a well known representation for schedules in which a horizontal open oblong is drawn against each activity indicating estimated duration. The tool can be adapted to any planner that writes plans in planning competition format to standard I/O. The user may choose the planner, the domain, and the problem file. Hence, the algorithm is fully planner independent. To avoid security conflicts in our web presentation of the visualizer we have incorporated a slightly different call panel that allows to select the planner in a pull-down menu and to include domain and problem by cut and paste.

The second program (suite) is *VepaDomain* for domain-dependent visualization of sequential plans. Figure 10 shows an example for the *Settlers* domain. *VepaDomain* includes instance independent visualizations for all competition domains all with less than 100 lines of code. The figures for the objects were collected with an image web search engine. We used *Google* (cf. `www.google.de`) in the advanced setting to search for small GIFs. *VepaDomain* works in cooperation with the MIPS system as follows. The planner writes propositional and numeric state facets and (unscheduled) action sequences into a file, which in turn is read by the domain dependent visualizer. Therefore, it is not difficult to adapt other planners to the domain visualizer.

8 Related Work

Solving planning problems with numerical preconditions and effects as allowed in Level 2 and Level 3 problems is undecidable in general (Helmert, 2002). However, the structures of the provided benchmark problems are simpler than the general problem class, so that these problems are in fact solvable.

8.1 Problem Classes and Methods

According to the PDDL-hierarchy we indicate three problem classes:

1. Propositional Planning. STRIPS problems have been tackled with different planning techniques, most notably by SAT-planning, e.g. (Kautz & Selman, 1996), IP-planning,

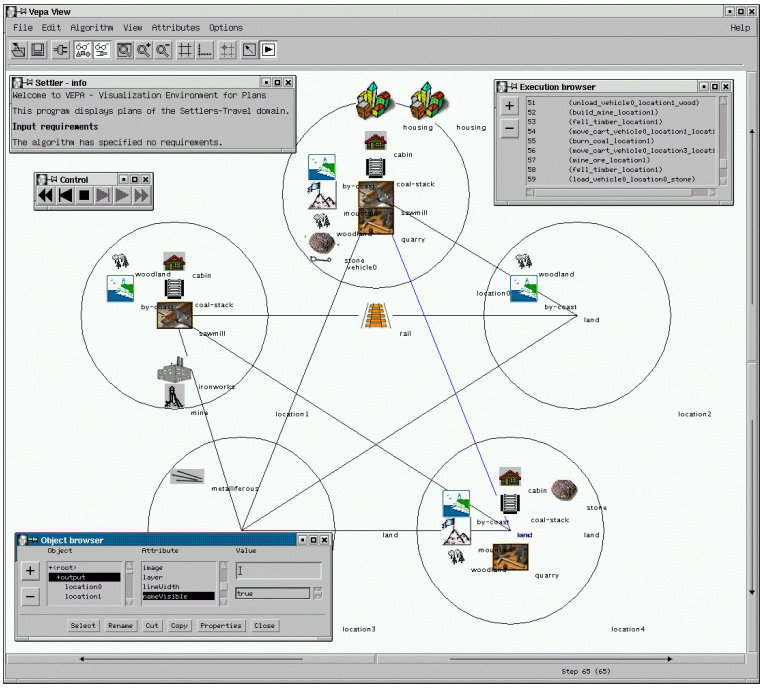

Figure 11: Visualization of a Planning Problem Instance of Settlers.

e.g. (Kautz & Walser, 1999), CSP-planning, e.g. (Rintanen & Jungholt, 1999), graph relaxation, e.g. (Blum & Furst, 1995), and heuristic search planing, e.g. (Bonet & Geffner, 2001). The major quality measurements are the numbers of sequential and parallel steps. ADL generalizations (Pednould, 1989) 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 can 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.

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 (Pettersson, 1999). 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 (Pettersson, 1999). 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. The absence of partitioning is current research (Wolper & Boigelot, 1998).

Critical path analysis for timed precedence networks is one of the simpler cases for scheduling. We have achieved a simplification by solving the sequential path problem first. Several scheduling techniques apply the presented critical path analyses as a subcomponent (Syslo, Deo, & Kowalik, 1983).

Most previously achieved results in symmetry reduction, e.g. (Guéré & Alami, 2001), neglect the combinatorial explosion problem and tend to assume that the information on existing symmetries in the domain is supplied by the user. Our approach shares similarities with the approach of (Fox & Long, 1999, 2002) in inferring symmetry information automatically, which bases on the TIM inference module (Fox & Long, 1998). Since no additional information on the current symmetry level in form of matrix is stored, our approach consumes less space per state. Moreover, we can give correctness proofs and efficiency guarantees.

8.2 Competing Planners

The on-line presentation of IPC-3[4] provides aspects of the input language, domains, results and other resources, e.g links to competing planners and the history of the event. In the following we briefly present the successful approaches at AIPS-2002. In AIPS-1998 most successful planners besides HSP (Bonet & Geffner, 2001) and *Satplan* (Kautz & Selman, 1996) were *Graphplan* derivates, e.g. IPP (Koehler et al., 1997) and STAN (Long & Fox, 1998). In 2000, the field was dominated by the success of heuristic search planning as in FF (Hoffmann & Nebel, 2001), HSP-2 (Haslum & Geffner, 2000), and in some hybrids, like STAN4 (Long & Fox, 2001b), and MIPS. System R (Lin, 2001) used backward regression.

Metric-FF (Hoffmann, 2002a) extends FF (Hoffmann & Nebel, 2001) and is a forward chaining heuristic state space planner. It performed best in the numerical track and was the only system besides MIPS that solved instances to *Settlers*. The main heuristic of relaxed plans bases on the HSP-heuristic (Bonet & Geffner, 2001). Metric-FF deals with PDDL 2.1 level 2, combined with ADL. The key difference is the definition of the relaxation. In STRIPS, the task is relaxed by ignoring all delete lists. However, numerical constraints are not monotonic: while one constraint (e.g. $x > 2$) might prefer higher values of a variable x, another constraint (e.g. $x < 2$) might prefer lower values. Opposed to that, the conditions in the purely logical case all prefer *higher* values of the propositional variables: negative conditions are compiled away as a pre-process, and thus it is always preferable to have more propositional facts true. The observation exploited in Metric-FF is that the same methodology can be applied in the numerical setting, at least in a subset of the language. The task is pre-processed such that all numerical constraints are monotonic, i.e., for any constraint c, if c is true in a state S then c is true in any state S' where, for all variables x, $x(S') \geq x(S)$. The relaxation is then simply to ignore all effects that decrease the value of the affected variable, and the relaxed task can be solved in Graphplan-style. To achieve the monotonicity property, one needs, in the numerical constraints and effects, expressions that are monotonic in all variables. In the current implementation, Numerical-FF restricts to linear expressions which obviously have this property.

LPG (Local search for Planning Graphs) (Gerevini & Serina, 2002) is the only planner that was competitive with MIPS at AIPS-2002 in the temporal domains. It bases on local search and planning graphs that handles PDDL 2.1 domains involving numerical quantities and durations. The system can solve both plan generation and plan adaptation problems. The basic search scheme of LPG was inspired by Walksat, an efficient procedure to solve SAT-problems. The search space of LPG consists of *action graphs* (Gerevini & Serina, 1999), particular subgraphs of the planning graph representing partial plans. The search steps are certain graph modifications transforming an action graph into another one. LPG exploits a compact representation of the planning graph to define the search neighborhood and to evaluate its elements using a parametrized function, where the parameters weight different types of inconsistencies in the current partial plan, and are dynamically evaluated during search using discrete Lagrange multipliers. The evaluation function uses some heuristics for estimate the *search cost* and the *execution cost* of achieving a (possibly numeric) precondition. Action durations and numerical quantities (e.g., fuel consumption) are represented in the actions graphs, and are modeled in the evaluation function. In temporal domains, actions are ordered using a *precedence graph* that is

[4] http://www.dur.ac.uk/d.p.long/competition.html

maintained during search, and that took into account the mutex relations of the planning graph.

TP4 (Haslum & Geffner, 2001) is in fact a scheduling system based on grounded problem instances. For these cases all formula trees in numerical conditions and assignments reduce to constants. Utilizing admissible heuristics TP4 minimize the plan objective of optimal parallel plan length. Our planner has some distinctive advantages: it handles numerical preconditions, instantiates numerical conditions on the fly and can cope with complex objective functions. Besides input restriction, in the competition, TP4 was somewhat limited by is focus to produce optimal solutions only.

SAPA (Do & Kambhampati, 2001) is a domain-independent time and resource planner that can cope with metrics and concurrent actions. It adapts the forward chaining algorithm of (Bacchus & Ady, 2001). Both planning approaches instantiate actions on the fly and can, therefore, in principle be adapted to at least mixed propositional and numerical planning problems. The search algorithm of SAPA extends partial concurrent plans. It uses a relaxed temporal planning graph for the yet unplanned events for different heuristic evaluation functions. In the competition SAPA was the only system besides MIPS that produced plans for the complex domains, which was the only one it submitted solutions to.

8.3 Symbolic Model Checking based Planners

In the 2000 competition, two other symbolic planner took part: PropPlan (Fourman, 2000), and BDDPlan (Hölldobler & Stör, 2000). Although they were not awarded for performance, they show interesting properties. PropPlan performs symbolic forward breadth first search to explore propositional planning problems with propositions for generalized action preconditions and generalized action effects. It performed well in the full ADL Micsonic-10 elevator domain (Koehler, 2000). ProbPlan is written in the Poly/ML implementation of SML and the standart C-BDD library[5]. BDD-Plan bases on solving the entailment problem in the fluent calculus with BDDs. At that time the authors acknowledged that a concise domain encoding and symbolic heuristic search as found in MIPS provides a large space for improvements.

In the Model-Based Planner, MBP[6], the paradigm of planning as symbolic model checking (Giunchiglia & Traverso, 1999) has been implemented for *non-deterministic* planning domains (Cimatti et al., 1998), which classifies in weak, strong, and strong-cyclic planning, with plans that are represented as complete state-action tables. For *partial observable* planning, exploration faces the space of belief states; the power set of the original planning space. Therefore, in contrast to the successor set generation based on action application, observations introduce "And" nodes into the search tree (Bertoli, Cimatti, Roveri, & Traverso, 2001b). Since the approach is a hybrid of symbolic representation of belief states and explicit search within the "And"-"Or" search tree, simple heuristic have been applied to guide the search. The need for heuristics that trade information gain for exploration effort is also apparent need in *conformant* planning (Bertoli, Cimatti, & Roveri, 2001a). Recent work (Bertoli & Cimatti, 2002) proposes improved heuristic for belief space planning. MBP has not yet participated in a planning competition, but plan to do in 2004.

The UMOP system parses a non-deterministic agent domain language that explicitly defines a controllable system in an uncontrollable environment (Jensen & Veloso, 2000). The planner also applies BDD refinement techniques such as automated transition function partitioning. New result for the UMOP system extends the setting of weak, strong and strong cyclic planning to adversarial planning, in which the environment actively influences the outcome of actions. In fact, the proposed algorithm joins aspects of both symbolic search and game playing. UMOP has not participated yet in a planning competition.

More recent developments in symbolic exploration are expected to influence automated planning in near future. With SetA*, (Jensen, Bryant, & Veloso, 2002) provide an improved imple-

[5] http://www-2.cs.cmu.edu/ modelcheck/bdd.html

[6] http://sra.itc.it/tools/mbp

mentation of the symbolic heuristic search algorithm BDDA* (Edelkamp & Reffel, 1998) and Weighted BDDA* (Edelkamp, 2001a). One major surplus is to maintain a finer granularity of the sets of states in the search horizon kept in a matrix according to matching g- and h- values. This contrasts the plain bucket representation of the priority queue based on f-values. The heuristic function is implicitly encoded with value differences of grounded actions. Since sets of states are to be evaluated and some heuristics are state rather than operator dependent it has still to be shown how general this approch is. As above the considered planning benchmarks are seemingly simple for single-state heuristic search exploration (Hoffmann, 2002b; Helmert, 2001). (Hansen, Zhou, & Feng, 2002) also re-implemented BDDA* and suggest that symbolic search heuristics and exploration algorithms are probably better to be implemented with algebraic decision diagrams (ADDs), as available in Somenzi's CUDD package. Although the authors achieved no improvement to (Edelkamp & Reffel, 1998) to solve the $(n^2 - 1)$-Puzzle, the established generalization to guide a symbolic version of the LAO* exploration algorithm (Hansen & Zilberstein, 2001) for *probabilistic* (MDP) planning, results in a remarkable improvement to the state-of-the-art (Feng & Hansen, 2002).

9 Conclusions

With the competing planning system MIPS, we have contributed an object-oriented architecture for a forward chaining, heuristic explicit and symbolic search planner that finds plans in finite-branching numerical problems. The planner parses, pre-compiles, solves, and schedules all current benchmark problem instances include complex ones with duration, resource variables and different objective functions.

Model checking aspects have always been influencing to the development of MIPS, e.g in the static analysis to minimize the state description length, in symbolic exploration and plan extraction, in the dependence relation for PERT schedules according to a given partial order, in bit-state hashing for IDA*, etc. The successes of planning with MIPS were also exported back to model checking, as the development of a heuristic search explicit-state model checker HSF-SPIN (Edelkamp et al., 2002) indicates.

MIPS instantiates numerical pre- and postconditions on-the-fly and produces optimized parallel plans. Essentially planning with numerical quantities and durative actions is planning with time and resources. The given framework of mixed propositional and numerical planning problems problems and the presented intermediate format can be seen as a normal form for temporal and metric planning.

For temporal planning, MIPS generates sequential (totally ordered) plans and efficiently schedules them with respect to the set of actions and the imposed causal structure, without falling into known NP-hardness traps for optimized partial-ordering of sequentially generated plan. For smaller problems the enumeration approach guarantees optimal solutions. To improve solution quality in approximate enumeration, the (numerical) estimate for the number of operators was substituted by scheduling the relaxed plan in each state.

Other contributions besides the new expressivity were refined static analysis techniques to simplify propositionally grounded representation and to minimize state encoding, automated state-based dynamic symmetry detection, as well as effective hashing and transposition cuts.

In the main part of paper we have analyzed completeness and optimality of different forms of exploration and have given a troughout theoretical treatment of PERT scheduling and symmetry detection, proving correctness results and studying run-time complexities.

References

Bacchus, F., & Ady, M. (2001). Planning with resources and concurrency: A forward chaning approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 417–

424.

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116*, 123–191.

Bäckström, C. (1998). Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, *9*, 99–137.

Bertoli, P., & Cimatti, A. (2002). Improving heuristics for planning as search in belief space. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Bertoli, P., Cimatti, A., & Roveri, M. (2001a). Heuristic search symbolic model checking = efficient conformant planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 467–472.

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2001b). Planning in nondeterministic domains under partial observability via symbolic model checking. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 473–478.

Biere, A. (1997). μcke - efficient μ-calculus model checking. In *Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science, pp. 468–471. Springer.

Blum, A., & Furst, M. L. (1995). Fast planning through planning graph analysis. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1636–1642.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, *24*(3), 142–170.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 165–204.

Cimatti, A., Giunchiglia, E., Giunchiglia, F., & Traverso, P. (1997). Planning via model checking: A decision procedure for AR. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 130–142. Springer.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *National Conference on Artificial Intelligence (AAAI)*, pp. 875–881.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. MIT Press.

Clarke, E. M., McMillan, K. L., Dill, D. L., & Hwang, L. J. (1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, *98*(2), 142–170.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. The MIT Press.

Dial, R. B. (1969). Shortest-path forest with topological ordering. *Communication of the ACM*, *12*(11), 632–633.

Do, M. B., & Kambhampati, S. (2001). Sapa: a domain-independent heuristic metric temporal planner. In *European Conference on Planning (ECP)*, pp. 109–120.

Edelkamp, S. (2001a). Directed symbolic exploration and its application to AI-planning. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, pp. 84–92.

Edelkamp, S. (2001b). First solutions to PDDL+ planning problems. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, pp. 75–88.

Edelkamp, S. (2001c). Planning with pattern databases. In *European Conference on Planning (ECP)*. 13-24.

Edelkamp, S. (2002a). Mixed propositional and numerical planning in the model checking integrated planning system. In *International Conference on AI Planning & Scheduling (AIPS), Workshop on Temporal Planning*.

Edelkamp, S. (2002b). Symbolic pattern databases in heuristic search planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Edelkamp, S., & Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 135–147. Springer.

Edelkamp, S., & Helmert, M. (2000). On the implementation of MIPS. In *Artificial Intelligence Planning and Scheduling (AIPS)–Workshop on Model Theoretic Approaches to Planning*, pp. 18–25.

Edelkamp, S., & Helmert, M. (2001). The model checking integrated planning system MIPS. *AI-Magazine*, 67–71.

Edelkamp, S., Leue, S., & Lluch-Lafuente, A. (2002). Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology (STTT)*.

Edelkamp, S., & Meyer, U. (2001). Theory and practice of time-space trade-offs in memory limited search. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science, pp. 169–184. Springer.

Edelkamp, S., & Reffel, F. (1998). OBDDs in heuristic search. In *German Conference on Artificial Intelligence (KI)*, pp. 81–92.

Edelkamp, S., & Reffel, F. (1999). Deterministic state space planning with BDDs. In *European Conference on Planning (ECP), Preprint*, pp. 381–382.

Edelkamp, S., & Stiegeler, P. (2002). Implementing HEAPSORT with $n \log n - 0.9n$ and QUICKSORT with $n \log n + 0.2n$ comparisons. *ACM Journal of Experimental Algorithmics*.

Feng, Z., & Hansen, E. (2002). Symbolic heuristic search for factored markov decision processes. In *National Conference on Artificial Intelligence (AAAI)*.

Fikes, R., & Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fourman, M. P. (2000). Propositional planning. In *Artificial Intelligence Planning and Scheduling (AIPS)-Workshop on Model-Theoretic Approaches to Planning*, pp. 10–17.

Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *Artificial Intelligence Research*, *9*, 367–421.

Fox, M., & Long, D. (1999). The detection and exploration of symmetry in planning problems. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 956–961.

Fox, M., & Long, D. (2001). PDDL2.1: An extension to PDDL for expressing temporal planning domains. Tech. rep., University of Durham, UK.

Fox, M., & Long, D. (2002). Extending the exploitation of symmetries in planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Gerevini, A., & Serina, I. (1999). Fast planning through greedy action graphs. In *National Conference of Artificial Intelligence (AAAI)*.

Gerevini, A., & Serina, I. (2002). LPG: a planner based on local search for planning graphs with action costs. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Giunchiglia, F., & Traverso, P. (1999). Planning as model checking. In *European Conference on Planning (ECP)*, pp. 1–19.

Guéré, E., & Alami, R. (2001). One action is enough to plan. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Hansen, E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*, 35–62.

Hansen, E. A., Zhou, R., & Feng, Z. (2002). Symbolic heuristic search using decision diagrams. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on on Systems Science and Cybernetics*, *4*, 100–107.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 140–149.

Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In *European Conference on Planning (ECP)*, pp. 121–132.

Helmert, M. (2001). On the complexity of planning in transportation domains. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 349–360. Springer.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Hipke, C. A. (2000). *Verteilte Visualisierung von Geometrischen Algorithmen*. Ph.D. thesis, University of Freiburg.

Hoffmann, J. (2000). A heuristic for domain independent planning and its use in an enforced hill climbing algorithm. In *ISMIS*, Lecture Notes in Computer Science, pp. 216–227. Springer.

Hoffmann, J. (2002a). Extending FF to numerical state variables. In *European Conference on Artificial Intelligence*.

Hoffmann, J. (2002b). Local search topology in planning benchmarks: A theoretical analysis. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Hoffmann, J., & Nebel, B. (2001). Fast plan generation through heuristic search. *Artificial Intelligence Research*, *14*, 253–302.

Hölldobler, S., & Stör, H.-P. (2000). Solving the entailment problem in the fluent calculus using binary decision diagrams. In *Artificial Intelligence Planning and Scheduling (AIPS)-Workshop on Model-Theoretic Approaches to Planning*, pp. 32–39.

Jensen, R. M., Bryant, R. E., & Veloso, M. M. (2002). SetA*: An efficient BDD-based heuristic search algorithm. In *National Conference on Artificial Intelligence (AAAI)*.

Jensen, R., & Veloso, M. M. (2000). OBDD-based universal planning for synchronized agents in non-deterministic domains. *Artificial Intelligence Research*, *13*, 189–226.

Kabanza, F., Barbeau, M., & St-Denis, R. (1997). Planning control rules for reachtive agents. *Artificial Intelligence*, *95*(1), 67–113.

Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI)*, pp. 1194–1201.

Kautz, H., & Walser, J. (1999). State-space planning by integer optimization. In *National Conference on Artificial Intelligence (AAAI)*.

Knoblock, C. (1994). Generating parallel execution plans with a partial order planner. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 98–103.

Koehler, J. (2000). Elevator control as planning problem. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 331–338.

Koehler, J., Nebel, B., & Dimopoulos, Y. (1997). Extending planning graphs to an adl subset. In *European Conference on Planning (ECP)*, pp. 273–285.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*(1), 97–109.

Lago, U. D., Pistore, M., & Traverso, P. (2002). Planning with a language for extended goals. In *National Conference on Artificial Intelligence (AAAI)*.

Lin, F. (2001). System r. *AI-Magazine*, 73–76.

Lind-Nielsen, J. (1999). Buddy: Binary decision diagram package, release 1.7. Technical Univeristy of Denmark. jln@itu.dk.

Long, D., & Fox, M. (1998). Efficient implementation of the plan graph in STAN. *Artificial Intelligence Research*, *10*, 87–115.

Long, D., & Fox, M. (2001a). Encoding temporal planning domains and validating temporal plans. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*.

Long, D., & Fox, M. (2001b). Hybrid stan: Identifying and managing combinatorial optimisation sub-problems in planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 445–452.

McDermott, D. (2000). The 1998 AI Planning Competition. *AI Magazine*, *21*(2).

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Press.

Pearl, J. (1985). *Heuristics*. Addison-Wesley.

Pednault, E. P. D. (1986). Formulating multiagend, dynamic-world problems in the classical framework. In *Reasoning about Action and Plans*, pp. 47–82. Morgan Kaufmann.

Pednould, E. (1989). ADL: Exploring the middleground between Strips and situation calculus. In *Knowledge Representation (KR)*, pp. 324–332. Morgan Kaufman.

Pettersson, P. (1999). *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. Ph.D. thesis, Department of Computer Systems, Uppsala University.

Pistore, M., & Traverso, P. (2001). Planning as model checking for extended goals in non-deterministic domains. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. *Machine Intelligence*, *8*, 55–72.

Reffel, F., & Edelkamp, S. (1999). Error detection with directed symbolic model checking. In *World Congress on Formal Methods (FM)*, pp. 195–211.

Regnier, P., & Fade, B. (1991). Détermination du parallélisme maximal et optimisation temporelle dans les plans d'actions linéaires. *Revue d'intelligence artificielle*, *5*(2), 67–88.

Reinefeld, A., & Marsland, T. (1994). Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*(7), 701–710.

Rintanen, J., & Jungholt, H. (1999). Numeric state variables in constraint-based planning. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 109–121. Springer.

Syslo, M. M., Deo, N., & Kowalik, J. S. (1983). *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall.

Veloso, M. M., Pérez, M. A., & Carbonell, J. G. (1990). Nonlinear planning with parallel resource allocation. In *Innovative Approaches to Planning, Scheduling and Control*, pp. 207–212.

Wolper, P., & Boigelot, B. (1998). Verifying systems with infinite but regular state spaces. In *Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pp. 88–97. Springer.