Numeric State Variables in Constraint-Based Planning

Jussi Rintanen[1] and Hartmut Jungholt[2]

[1] Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Am Flughafen 17, 79110 Freiburg im Breisgau
Germany
[2] Universität Ulm
Fakultät für Informatik
Albert-Einstein-Allee, 89069 Ulm
Germany

Abstract. We extend a planning algorithm to cover simple forms of arithmetics. The operator preconditions can refer to the values of numeric variables and the operator postconditions can modify the values of numeric variables. The basis planning algorithm is based on techniques from propositional satisfiability testing and does not restrict to forward or backward chaining. When several operations affect a numeric variable by increasing and decreasing its value in parallel, the effects have to be combined in a meaningful way. This problem is especially acute in planning algorithms that maintain an incomplete state description of every time point of a plan execution. The approach we take requires that for operators that are executed in parallel, all linearizations of the operations to total orders behave equivalently. We provide an efficient and general solution to the problem.

1 Introduction

In automated planning and for example in generating counterexamples in verification of safety properties of transition systems, the goal is to find a sequence of state transitions that lead from a given initial state to a state that satisfies certain properties. There are many possible ways of describing transition systems. The most basic description used in automated planning is based on operators that are applicable when a number of Boolean state variables are true or false and that make a number of Boolean state variables true or false. Many transition systems can be described by using this kind of operators. However, the descriptions would be more concise if no restriction to two-valued Boolean state variables were made. State variables with n possible values can always be described with $\lceil \log_2 n \rceil$ Boolean state variables. However, replacing an operator description that refers to many-valued variables by ones that refer to Boolean variables only leads to a big increase in the number of operators. This reduces the efficiency of planning algorithms, and is therefore often not feasible.

Apart from the practical problems in reducing multi-valued state variables to Boolean state variables, there are semantic problems related to parallelism that are not addressed by that reduction. It is often the case that parallel operations affect the same numeric variables, for example the amount of money. The reductive approach to multi-valued variables in this case is not applicable because operations affecting the same variables that other parallel operations refer to in their preconditions, are not allowed. Therefore parallel operations have to be given a semantics that is aware of numeric variables and operations on them. For example, it should be possible to simultaneously execute a number of operations that increase and decrease the amount of money so that under the semantics the result is well-defined.

In this paper we investigate implementing parallel operations in a general planning framework based on constraint propagation. The operators may increase or decrease state variables with a fixed amount, or assign a fixed value to a state variable. Despite the restricted form of changes we consider, the work gives a solution to the problem of parallelism also for more general cases, for example when state variables may be assigned new values that are complex expressions that refer to constants and other numeric state variables.

2 Numeric State Variables in Operator Definitions

We extend the notion of operators with state variables that may take numeric values. Numeric state variables may be referred to in operator preconditions, and operators' postconditions may change the values of the state variables. The basic problems inherent in handling numeric state variables can be demonstrated with integers and constant increments and decrements. Preconditions of operators may include expressions $R = [n..m]$ which requires that $R \geq n$ and $R \leq m$. Postconditions of operators may be assignments $R := n$, $R := R + n$ and $R := R - n$ that respectively assign a specific value to R or increase or decrease the value of R by n. We do not allow disjunctions of preconditions $R = [n..m]$, as this would complicate the reasoning we have to perform. Conjunctions of preconditions $R = [n..m]$ are equivalent to intervals that are intersections of the conjunct intervals. Unbounded intervals can be formed with the constant infinite ∞ as $[-\infty..n]$ and $[n..\infty]$. The sum $\infty + n$ for an integer n is ∞. Notice that ordinary Boolean state variables can be understood as numeric variables, for example by interpreting the interval $[0..0]$ as false and the interval $[1..1]$ as true.

3 Parallelism

Most conventional planners that allow parallelism, including planners of the partial-order planning paradigm [7] and more recent planning algorithms [1], restrict to cases where two parallel operations may not have variables in common in their postconditions, and variables occurring in the postconditions of one may not occur in the preconditions of the other. Two operators related in this way are said to be *dependent* because executing one may disable or enable the execution

of the other, or falsify the effects of the other. The purpose of these restrictions is to guarantee a well-defined meaning for parallelism: parallelism means the possibility of interleaving the operators to a sequence in all possible ways, and the effect of executing the operators has to be the same for all interleavings.

In many applications, for example when the variables represent money or some physical resources, the requirements on parallel operators stated earlier are too strict, and in some applications may prevent parallelism altogether. For example, if all operators are associated with a cost and no two operators increasing the cost may be executed in parallel, no parallelism is possible.

Therefore we decided to try to relax these requirements. It turns out that interesting forms of parallelism, not considered in earlier work in planning, are possible. A set of operators executed in parallel have to fulfill the requirement used in earlier work on planning: parallelism is allowed as far as the execution of a set of operators in all possible interleavings produces the same result. Instead of using the syntactic condition used in connection with Boolean state variables, we develop a more abstract criterion parallel operations have to fulfill, and show that it can be implemented efficiently. It is not obvious how this can be done.

Example 1. Consider the operators $R = [1..2] \Rightarrow R := R + 2$ and $R = [1..5] \Rightarrow R := R + 2$. Initially $R = 1$. The operators can be interleaved in two ways. In the first interleaving the first operator is executed first, the value of R is increased by 2 to 3, and then the second operator is executed, R getting the value 5. However, the second interleaving does not lead to a legal sequence of operations. First the second operator is applied, increasing the value of R to 3. Now the precondition of the first operator is not true, and the interleaving does not represent a valid execution. Hence the operators cannot be executed in parallel.

4 The Basis Algorithm

To investigate the problem of parallel updates of numeric state variables in practise, we decided to implement our techniques in an existing automated planner. The reason for extending the algorithm presented by Rintanen [8] to handle numeric state variables is that the algorithm subsumes both forward-chaining and backward-chaining, as the algorithm can simulate them by making branching decisions in a temporally directed manner starting from the goal state and proceeding towards the initial state, or vice versa. The algorithm by Rintanen is motivated by the planning as satisfiability approach to classical planning [3].

Each run of the main procedure of the algorithm is parameterized by the plan length. The main procedure is called with increased plan lengths 1, 2, 3, 4 and so on, until a plan is found. The algorithm maintains a state description for each time point of the plan execution. This description is incomplete in the sense that not all truth values of state variables need to be known. Initially, it includes the unique initial state and a description of the goal states.

For a given plan length and initial and goal states, planning proceeds by binary search. The algorithm chooses an operator o and a time point t, and

records in its data structures that operator o is applied at t (or respectively that is not applied at t.) Each decision to apply or not apply is followed by reasoning concerning the new situation. For example for an applied operator its preconditions are true at t and postconditions are true at $t+1$ and dependent operators cannot be applied at t. The decision to not apply a certain operator at t may indicate that a certain state variable retains its truth-value between t and $t+1$, thereby allowing to infer the value of the variable at t if its value at $t+1$ was known, and vice versa. A more detailed description of the constraint reasoning involved has been published elsewhere [8]. Upon detecting the inexistence of plans in the current subtree, the algorithm backtracks.

Figure 1 shows a part of a search tree produced by the algorithm. Oi represents a particular operator application (an operator and a time point), and N-Oi that the operator application is not performed. Each arrow corresponds to a decision to (not) apply a certain operator at a certain point of time, and inferred applications and inapplications of other operations. For example, in the first subtree of the first child node of the root, choosing to not apply O2 leads to inferring the operation O3.

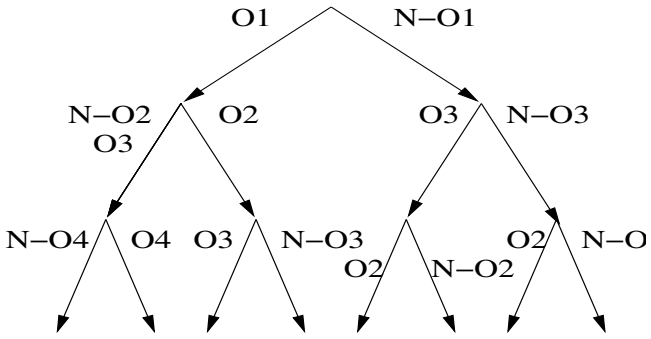

Fig. 1. Part of a search tree of the planning algorithm

When reaching a leaf node in the search tree without detecting any inconsistencies, that is, we have committed to the application and non-application of all operators at all time points, a plan has been found.

An important technique discovered in the context of algorithms for propositional satisfiability is detection of failed literals by unit resolution [6]. This technique in the setting of our algorithm proceeds as follows. An operation is labeled APPLIED (respectively NOT-APPLIED), and the inferences directly following from this are made. If an inconsistency is found, the operator must be labeled NOT-APPLIED instead (respectively APPLIED.) This technique is very useful because it allows to infer operator application and inapplications by inexpensive (low polynomial time) computation, in many cases leading to a big reduction in the depth of the search tree. A linear reduction in the search tree

depth corresponds to an exponential reduction in the number of nodes in the search tree and consequently in the computation time.

5 Extension to the Basis Algorithm

The initial state of a problem instance specifies the values of all numeric state variables uniquely, and the goal specifies values of some numeric state variables as an interval. When the main program of the planner is run, all the processing of numeric state variables goes through a single function call that indicates that the main program has decided to apply or not to apply a certain operator. In addition, there are functions for registering a backtracking point and for undoing all the changes done since the previous backtracking point was registered.

From the point of view of numeric state variables, the algorithmic question that has to be solved is the following. Given a sequence of operators, labeled APPLIED or NOT-APPLIED (corresponding to a directed path starting from the root of the tree in Figure 1), decide whether this information is consistent with the initial values of the variables, with the values determined by the goal, and with the intermediate values inferred for the variables. Consistency here means that there is a plan execution (and hence also a plan) that executes operations labeled APPLIED and does not execute operations labeled NOT-APPLIED. Executing an operator means that its preconditions are true and the changes that take place exactly correspond to the changes caused by the operations that are executed.

For the correctness of the planning algorithm, the constraint reasoner has to satisfy the following two requirements.

1. When all operator applications have been labeled, an inconsistency must be reported if there are no executions where exactly the APPLIED operations are performed.
2. If an inconsistency is reported, there may not be executions that apply operations labeled APPLIED and do not apply operations labeled NOT-APPLIED.

These requirements for example allow refraining from doing any constraint reasoning as long as some operation applications are not labeled. This strategy, even though correct, would not be very efficient as there are usually a lot of possibilities of detecting inconsistencies much earlier and to avoid traversing parts of the search tree that do not contain plans.

For simplicity of presentation, in the following we discuss the case with only one numeric state variable R. Inferences performed for different numeric variables do not interact and are done in complete separation. The value of R at a time point t is represented by R_t.

The two kinds of changes in state variables are handled differently. Assignments $R := n$ at t are simply performed by assigning $R_t := [n..n]$. Assignments obviously cannot be performed in parallel with other assignments or increments

or decrements because different linearizations would give different results. The more complicated case is that of increments and decrements.

For reasoning about the values of R efficiently, we introduce the following auxiliary variables.

M_t^I Maximum possible increase at time t. This is the sum of the increases of operators that are not labeled NOT-APPLIED at t, minus the decreases of operators that are labeled APPLIED at t.

M_t^D Maximum possible decrease at time t. This is symmetrically the sum of the decreases of operators that are not labeled NOT-APPLIED at t, minus the increases of operators that are labeled APPLIED at t.

I_t^I Maximum intermediate increase at time t. This is the sum of increases of operators labeled APPLIED at t. The variable is needed for the test that the preconditions are satisfied under all interleavings.

I_t^D Maximum intermediate decrease at time t. This is the sum of decreases of operators labeled APPLIED at t.

When all operator applications have been labeled, $M_t^I = -M_t^D = I_t^I - I_t^D$ for all t. The values of these auxiliary variables can be recomputed every time they are needed. Alternatively, and more efficiently, they can be incrementally maintained when operations get labels. Initially, M_t^I is assigned the sum of increments of R in postconditions of all operators for all t, M_t^D similarly the sum of decrements of R, and I_t^I and I_t^D are assigned 0.

When an operator that increases R by n is labeled APPLIED at t, set $M_t^D := M_t^D - n$ and $I_t^I := I_t^I + n$.

When an operator that decreases R by n is labeled APPLIED at t, set $M_t^I := M_t^I - n$ and $I_t^D := I_t^D + n$.

When an operator that increases R by n is labeled NOT-APPLIED at t, set $M_t^I := M_t^I - n$.

When an operator that decreases R by n is labeled NOT-APPLIED at t, set $M_t^D := M_t^D - n$.

The following rules for propagating constraints on neighboring values of R are needed. The first two are applied only when all operators making constant-value assignments to R at t are labeled NOT-APPLIED. The rules are applied whenever some of the variables involved change.

1. If $R_t = [a..b]$, then $R_{t+1} := R_{t+1} \cap [a - M_t^D..b + M_t^I]$.
2. If $R_{t+1} = [a..b]$, then $R_t := R_t \cap [a - M_t^I..b + M_t^D]$.
3. If an operator with precondition $R = [a..b]$ is applied at t, then $R_t := R_t \cap [a..b]$.

When a number of operators are applied in parallel at t, it is not sufficient to test that the preconditions of the operators are satisfied at t. It is also necessary to test that the preconditions are satisfied in all intermediate states of executions of all linearizations of the operators. The standard interpretation of parallelism requires this: all interleavings/linearizations have to be possible and the result of executing the operations has to be the same in all cases.

The test that the executions of all linearizations are possible is based on the auxiliary variables I_t^I and I_t^D that indicate how high and how low the value of the state variable can get when performing the execution of the operators between t and $t+1$.

Assume that $R = [a..b]$ at t. Now the value of R can get as high as $a + I_t^I$ under some linearization of the operators executed between t and $t+1$ (and higher if the value turns out to be higher than a, for example b), and as low as $b - I_t^D$. We have to test that the preconditions of none of the operators are violated under any of the linearizations.

Consider an operator o with the precondition $R = [p..q]$ that does not affect the value of R. Now if $b - I_t^D < p$, then there is a linearization of the operators during the execution of which the precondition of o is not fulfilled: execute first all operators that decrease the value of R. Similarly, if $a + I_t^I > q$, we have detected an inconsistency.

In the general case the operator o may have a precondition $R = [p..q]$ and also affect the value of R. Because the change caused by o cannot affect the fulfillment of its own precondition, the effect of o on the value of R has to be ignored when testing the precondition of o with respect to all interleavings. If o increases R by n, define $n^+ = n$ and $n^- = 0$. If o decreases R by n, define $n^+ = 0$ and $n^- = n$. Now we can state the test for operator preconditions under parallelism in the general case. Assume that the value of R at t is $[a..b]$. If $b - (I_t^D - n^-) < p$, we have detected an inconsistency. Similarly, if $a + (I_t^I - n^+) > q$, we have detected an inconsistency. These checks are subsumed by the following constraint on the value of R.

$$R_t := R_t \cap [p + (I_t^D - n^-)..q - (I_t^I - n^+)]$$

This constraint stems from the possibility of executing all other operators that increase (or decrease) R before executing o. This way the intermediate value of R might violate the precondition of o. To avoid this violation, the initial value of R has to satisfy the constraint.

The correctness of the linearization test may not be obvious, for example whether the test allows all those sets of parallel operations that can be executed in any order with the same final result. Assume that no execution violates the precondition of a particular operator. Then we have to be able to first execute all decrementing (incrementing) operators and the particular operator immediately afterwards so that the lower-bound (upper-bound) of the precondition is not violated. This is exactly the test we perform.

Example 2. Consider the operators $R = [1..5] \Rightarrow R := R + 2$ and $R = [4..6] \Rightarrow R := R - 2$ that are applied in parallel at t. From the preconditions we directly get the constraint $R_t = [4..5]$. From the precondition of the first operator under the interleaving consideration we get $R_t := R_t \cap [1 + (2 - 0)..5 - (2 - 2)]$. From the second operator we get $R_t := R_t \cap [4 + (2 - 2)..6 - (2 - 0)]$. Hence $R_t := [4..5] \cap [3..5] \cap [4..4] = [4..4]$, and 4 is the only possible value for R at t: otherwise there would be an interleaving in which one of the preconditions is violated.

For example for the initial value $R = 5$, executing the first operator makes the precondition of the second false.

We illustrate the propagation of constraints and the treatment of parallel operators with two further examples.

Example 3. Consider two operators that may respectively increase the value of R by 5 and decrease the value of R by 2. Initially the value of R is 0, and at time point 3 it is 1. We construct a sequence of operations that take us from the initial value 0 to the goal value 1.

The computation of the values of R at $t \in \{1, 2\}$ is based on the values of R at 0 and 3 and the values of M_t^I and M_t^D. For example, the value of R_1 is the intersection of $[0 - M_0^D..0 + M_0^I]$ and $[1 - M_2^I - M_1^I..1 + M_2^D + M_1^D]$, that is $[-2..5] \cap [-9..5] = [-2..5]$. (See the leftmost table below.) Now we apply op-2 at 1 and commit to not applying op5 at 1. (See the rightmost table below.)

t	0	1	2	3
R_t	[0..0]	[−2..5]	[−4..3]	[1..1]
M_t^I	5	5	5	
M_t^D	2	2	2	
I_t^I	0	0	0	
I_t^D	0	0	0	
op5				
op-2				

t	0	1	2	3
R_t	[0..0]	[−2..5]	[−4..3]	[1..1]
M_t^I	5	-2	5	
M_t^D	2	2	2	
I_t^I	0	0	0	
I_t^D	0	2	0	
op5		F		
op-2		T		

And we apply op5 at 2 and commit to not applying op-2 at 1. (See the leftmost table below.) And finally, the only possibility left is to apply op-2 at 0. (See the rightmost table below.)

t	0	1	2	3
R_t	[0..0]	[−2.. − 2]	[−4.. − 4]	[1..1]
M_t^I	5	-2	5	
M_t^D	2	2	-5	
I_t^I	0	0	5	
I_t^D	0	2	0	
op5		F	T	
op-2		T	F	

t	0	1	2	3
R_t	[0..0]	[−2.. − 2]	[−4.. − 4]	[1..1]
M_t^I	-2	-2	5	
M_t^D	2	2	-5	
I_t^I	0	0	5	
I_t^D	2	2	0	
op5	F	F	T	
op-2	T	T	F	

Next we consider an example in which it is essential to check the interleavings.

Example 4. There are three operations, get5DM, lose2DM and lose4DM that respectively change the value of R by 5, -2 and -4. The initial value of R is 5. The decrementing operations have as their precondition that the value of R is respectively at least 2 and at least 4.

This scenario could be viewed as transactions on somebody's bank account.

We now try to schedule one instance of each of these operations on two points of time that we call Monday and Tuesday (there may be Boolean variables and a goal – invisible to the constraint reasoner – that enforces this task.) The initial

situation is as shown below on the left. Then we try to see whether it is possible to apply both lose2DM and lose4DM on Monday (and not on Tuesday.) Before applying the linearization constraints the situation is as shown below on the right.

t	Mon	Tue	Wed
R_t	[5..5]	[−1..10]	[−7..15]
M_t^I	5	5	
M_t^D	6	6	
I_t^I	0	0	
I_t^D	0	0	
get5DM			
lose2DM			
lose4DM			

t	Mon	Tue	Wed
R_t	[5..5]	[−1..4]	[−1..9]
M_t^I	-1	5	
M_t^D	6	0	
I_t^I	0	0	
I_t^D	6	0	
get5DM			
lose2DM	T	F	
lose4DM	T	F	

The preconditions of both lose2DM and lose4DM are true on Monday and it still seems possible to achieve a positive balance for Tuesday as the interval upper bound is 4. However, we have to apply the rules $R_{\text{Mon}} := R_{\text{Mon}} \cap [2 + (I^D_{\text{Mon}} - 2)..\infty - (I^I_{\text{Mon}} - 0)] = R_{\text{Mon}} \cap [6..\infty]$ (imposed by lose2DM), and $R_{\text{Mon}} := R_{\text{Mon}} \cap [4 + (I^D_{\text{Mon}} - 4)..\infty - (I^I_{\text{Mon}} - 0)] = R_{\text{Mon}} \cap [6..\infty]$ (imposed by lose4DM). These both violate $R_{\text{Mon}} = [5..5]$ and therefore indicate that whichever operator is applied first, the precondition of the other would not be fulfilled.

Therefore we lose2DM and get5DM on Monday and lose4DM on Tuesday.

t	Mon	Tue	Wed
R_t	[5..5]	[8..8]	[4..4]
M_t^I	3	-4	
M_t^D	-3	4	
I_t^I	5	0	
I_t^D	2	4	
get5DM	T	F	
lose2DM	T	F	
lose4DM	F	T	

6 Experiments

To evaluate the efficiency of the planner, we ran it on a number of benchmarks earlier used in connection with another planner that handles numeric state variables [11] and extensions of some benchmarks used in connection with classical planners [2]. Our current implementation handles only increments and decrements in operator effects. Therefore in the benchmarks of Wolfman and Weld [11] we replaced constant assignments (fill the tank of an airplane or a truck) by increments (if the tank is less than half full, add fuel half the capacity.) We do not expect this modification to affect the runtimes significantly.

In Table 1 we give runtimes for the logistics benchmarks. These are numeric versions of the well-known logistics benchmarks for classical planners. The runtimes inside parentheses are for the computation after the plan length has been

determined. The total runtimes are given outside parentheses; in the LPSAT case it is the sum of the runtimes of finding the plan (of length n) and showing that plans of length $n-1$ do not exist [10]. Wolfman and Weld ran their program LPSAT on a 450 MHz Pentium II that is probably slightly faster than the 296 MHz Sun Ultra we used. The number of non-leaf search tree nodes is for our planner.

Table 1. Runtimes of four benchmarks on LPSAT and on our planner (in seconds)

problem	LPSAT		we		nodes
log-a	20.35	(12.1)	11.8	(6.0)	47
log-b	591.2	(576)	65.7	(23.1)	80
log-c	849.7	(830)	66.3	(23.7)	73
log-d	> 3600	(> 3600)	227.6	(140.5)	477

The logistics domain is very sensitive to the criteria according to which branching variables are chosen. The benchmarks are solved efficiently because the branching heuristic reliably guides the search to a plan needing no or very little backtracking. An earlier version of our planner that did not perform all possible inferences when evaluating branching variables and did not consider numeric state variables in the branching heuristic was not able to solve these benchmarks. The heuristic made early a wrong branching decision and this was discovered only after going 20 or 30 nodes deep in the search tree. Finding the way out by exhaustive search was not possible in a reasonable amount of time.

We also ran benchmarks from the trains domain of Dimopoulos et al. [2]. The numeric versions of these benchmarks add constrains on the total number of trips between cities. We determined the smallest possible numbers of trips with which these benchmarks remain solvable, and forced the planner to find such solutions. Both planners were run with post-serializable operations [2] which means that parallelism for Boolean variables was allowed as long as there is one linearization of the operators.

We ran three variants of this benchmark and give the results in Table 2. The first runtimes are with numeric state variables and the second runtimes for the original non-numeric versions. The runtimes in parentheses are for finding the plan when the length is known. The total runtimes are given outside parentheses.

Table 2. Runtimes of three benchmarks on our planners (in seconds)

problem	numeric		nodes	non-numeric		nodes
train-a10	13.5	(9.7)	17	13.0	(9.0)	15
train-b12	69.7	(58.7)	14	81.0	(69.5)	17
train-c13	153.8	(103.8)	27	180.4	(124.7)	31

7 Related Work

Koehler [5] extends the Graphplan algorithm [1] to handle numeric state variables. The numeric preconditions and effects Koehler considers are more general than the ones considered by us. She, for example, allows multiplication and division of resource values by constants and linear equations involving two resource values. The main difference to our work is that Koehler allows only a restricted amount of parallelism: two operators, one of which increases and the other decreases a numeric state variable, cannot be executed in parallel. Parallel operations are often crucial in achieving efficient planning because for n independent operators not all $n!$ linearizations have to be considered separately. The Graphplan framework restricts to backward-chaining and does not have the generality satisfiability planning or constraint-based planning have.

Kautz and Walser [4] show how integer programming can be a basis for efficient domain-independent planning with numeric state variables. Their framework makes it possible to directly minimize – for a fixed plan length – certain numeric values, for example the number of operators in the plan or the maximum value of a numeric state variable during a plan execution. Kautz and Walser also consider parallel operations like we do. Their main results are based on a local search algorithm that is not capable of determining the inexistence of plans satisfying certain properties; for example that there are no plans of certain length. Our planning algorithm systematically goes through the search space representing all possible plans and is therefore capable of determining the inexistence of plans having a certain property.

Vossen et al. [9] do classical planning by translating problem instances to integer programming. They use a standard integer programming solver and still are able to show that integer programming provides a solution method for classical planning that approaches the efficiency of general-purpose satisfiability algorithms on the same problems. There is the obvious possibility to extend the translations to cover numeric state variables. This is however not discussed further by Vossen et al.

Wolfman and Weld [11] present LPSAT, a combination of decision procedures for propositional satisfiability and linear programming. For satisfiability they use a variant of the Davis-Putnam procedure, and for linear programming an implementation of the Simplex algorithm. The algorithm used by Wolfman and Weld is systematic and is therefore capable of reporting the inexistence of solutions.

8 Conclusions

Obvious extensions to our framework of numeric variables are more complex operator preconditions and more complex effects of operators. In this work we have restricted to very simple operator preconditions to make it possible to represent incompletely known values of numeric variables as intervals. For example disjunctive preconditions would require unions of intervals, and this leads to

more complex constraint reasoning. Also more complex updates make the constraint reasoning more complicated. However, the problems of parallel updates are present in the current framework in their full extent, and we have presented solutions to these problems. For incorporating more complex updates in the current framework, techniques from reasoning with more complex arithmetic expressions could be directly applied.

Acknowledgements

This research was funded by the Deutsche Forschungsgemeinschaft SFB 527 and carried out while the first author was at the University of Ulm.

References

1. Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.
2. Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the Fourth European Conference on Planning (ECP'97)*, pages 169–181. Springer-Verlag, September 1997.
3. Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California, August 1996. AAAI Press / The MIT Press.
4. Henry Kautz and Joachim Walser. State-space planning by integer optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 526–533, 1999.
5. Jana Koehler. Planning under resource constraints. In *Proceedings of the 13th European Conference on Artificial Intelligence*, pages 489–493. John Wiley & Sons, 1998.
6. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 366–371, Nagoya, Japan, August 1997.
7. David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In T. L. Dean and K. McKeown, editors, *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 634–639, Anaheim, California, 1991. The MIT Press.
8. Jussi Rintanen. A planning algorithm not based on directional search. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 617–624, Trento, Italy, June 1998. Morgan Kaufmann Publishers.
9. Thomas Vossen, Michael Ball, Amnon Lotem, and Dana Nau. On the use of integer programming models in AI planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume I, pages 304–309, Stockholm, 1999. Morgan Kaufmann Publishers.
10. Steven A. Wolfman, August 1999. email correspondence.
11. Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume I, pages 310–315, Stockholm, 1999. Morgan Kaufmann Publishers.

Appendix: Sample Output from the Planner

The following is output from the planner on a simplified logistics problem. The output is after the planner has reached plan length 7 and has not yet performed any search.

```
                                                01234567
                          at(bostruck,bosairpo) F   TTFF
                             at(bostruck,bospo) T   FFTT
                                at(p1,bosairpo) FFFFTFFF
                                   at(p1,bospo) FFFFFFFT
                                 at(p1,laairpo) TTFFFFFF
                                    at(p1,lapo) FFFFFFFF
                               at(plane1,bosairpo) TFFTT
                                at(plane1,laairpo) FTTFF
                                   in(p1,bostruck) FFFFFTTF
                                     in(p1,plane1) FFTTFFFF
   drive-truck(bostruck,bosairpo,bospo,bos) .  ..T.
   drive-truck(bostruck,bospo,bosairpo,bos)     ...
                         fly-plane-bos-la(plane1) T...
                         fly-plane-la-bos(plane1) ..T..
                    load-plane(plane1,p1,bosairpo) .......
                     load-plane(plane1,p1,laairpo) .T.....
                  load-truck(bostruck,p1,bosairpo) ....T..
                     load-truck(bostruck,p1,bospo) .......
                            refuel-plane(bosairpo) ...
                             refuel-plane(laairpo) .....
                   refuel-truck(bostruck,bosairpo) ...  ..
                      refuel-truck(bostruck,bospo) .......
                  unload-plane(plane1,p1,bosairpo) ...T...
                   unload-plane(plane1,p1,laairpo) .......
                unload-truck(bostruck,p1,bosairpo) .......
                   unload-truck(bostruck,p1,bospo) ......T
```

Variable	0	1	2	3	4	5	6
bostruck.fuel	100_ 100	80_ 100	40_ 100	0_ 100	-20_ 160	20_ 220	0_ 200
mpi:	0	0	0	60	60	-20	0
mpd:	20	40	40	20	0	20	0
ami:	0	0	0	0	0	0	0
amd:	0	0	0	0	0	20	0
plane1.fuel	400_ 400	250_ 250	250_ 250	100_ 100	100_ 300	-50_ 500	-350_ 900
mpi:	-150	0	-150	200	200	400	400
mpd:	150	0	150	0	150	300	300
ami:	0	0	0	0	0	0	0
amd:	150	0	150	0	0	0	0