

SAT Encodings of State-Space Reachability Problems in Numeric Domains

Jörg Hoffmann
DERI Innsbruck
Innsbruck, Austria
joerg.hoffmann@deri.org

Carla Gomes & Bart Selman
Cornell University
Ithaca, NY, USA
gomes|selman@cs.cornell.edu

Henry Kautz
University of Rochester
Rochester, NY, USA
kautz@cs.rochester.edu

Abstract

Translation to Boolean satisfiability is an important approach for solving state-space reachability problems that arise in planning and verification. Many important problems, however, involve numeric variables; for example, C programs or planning with resources. Focussing on planning, we propose a method for translating such problems into propositional SAT, based on an approximation of reachable variable domains. We compare to a more direct translation into “SAT modulo theory” (SMT), that is, SAT extended with numeric variables and arithmetic constraints. Though translation to SAT generates much larger formulas, we show that it typically outperforms translation to SMT almost up to the point where the formulas don’t fit into memory any longer. We also show that, even though our planner is optimal, it tends to outperform state-of-the-art sub-optimal heuristic planners in domains with tightly constrained resources. Finally we present encouraging initial results on applying the approach to model checking.

1 Introduction

Satisfiability testing is a significant method for solving state-space reachability problems. In 2004 and 2006, the winners of the track for optimal planners at the international planning competition translated bounded-length planning to SAT testing. In model checking, determining by SAT testing if or if not error states are reachable in a fixed number of steps often succeeds where BDD methods fail [Clarke *et al.*, 2001].

Many important planning and verification problems, however, naturally involve numeric variables and constraints. For example, C programs, or planning with operators that consume resources. The question arises as to how to best generalize the “encode as SAT” approach to numeric problems. So far, two different methods have been pursued:

- (A) Interpret numbers as bitstrings, and numeric operations as bitwise operations. “Implement” these operations in a propositional CNF.
- (B) Translate not to Boolean satisfiability but to “SAT modulo theory” (SMT), that is, SAT extended with numeric variables and arithmetic constraints.

In software verification, both method (A) and method (B) have been explored, for example by Clarke *et al* [2004] and Ganzinger *et al* [2004], respectively. In planning, so far only method (B) has been explored, namely in LPSAT [Wolfman and Weld, 1999] and TM-LPSAT [Shin and Davis, 2005].

The disadvantage of method (A) is that it involves the cumbersome implementation of bitwise operations in a CNF. The disadvantage of method (B) is that the expressivity of SMT languages comes at a price: the solvers are much more complex than SAT solvers, and do generally not scale as well. Herein, we explore a third option:

- (C) Approximate the *achievable domains*, i.e., determine sets $D_t(v)$ so that, for every variable v , every value that v can have after t transitions is contained in $D_t(v)$. These finite domains can then serve as the basis for a Boolean encoding, where atoms represent numeric variables taking on particular values.

Of course, this method will only work if the sets $D_t(v)$ do not grow too large. The underlying intuition is that the actual number of distinct values a numeric variable can achieve in a bounded-length problem is often quite small. Our experimental results confirm this.¹

We focus on planning. We approximate reachable variable domains by constructing a “numeric relaxed planning graph” (NRPG) inspired from work on generating heuristic functions [Hoffmann, 2003; Kupferschmid *et al.*, 2006]. We add some more intelligence in order to obtain smaller $D_t(v)$. Based on the NRPG we obtain propositional CNFs in a manner inspired by Kautz and Selman [1999]. The CNFs are handed over to the state-of-the-art SAT solver MiniSat [Een and Sörensson, 2003]. The plan length bound starts at 1 and is incremented until the first satisfiable formula is found – this way, the generated plans are optimal (have minimal length). We call the resulting planning system NumReach/SAT.

We run large-scale experiments on a broad variety of planning problems. We could not compare to the existing planning systems using method (B) since LPSAT is outdated and TM-LPSAT is not available (see also Section 4). So, for this comparison, we implemented such a system ourselves, called

¹There is related work [Seshia and Bryant, 2004] using a notion of relevant variable domains in solving Presburger formulas. The context and techniques are quite different from ours; e.g., the variable domains arise through an iterated interaction with a SAT solver.

NumReach/SMT.² This is *idempotent* to NumReach/SAT except for the encoding of numeric variables. The SMT formulas are handed over to the state-of-the-art SMT solver MathSat [Bozzano *et al.*, 2006]. NumReach/SAT consistently outperforms NumReach/SMT almost up to the point at which the (larger) SAT encodings no longer fit into memory.

We also compare our planners to the “sub-optimal heuristic planner” family, namely to Metric-FF [Hoffmann, 2003], LPG [Gerevini *et al.*, 2003], and SGPlan [Chen *et al.*, 2006]. These planners are fast (they respectively won the competitions 2000–2006) but are usually not compared to optimal planners since such a comparison is unfair. However, we find that, in contrast to the usual competition results, in our context optimal planners are quite competitive. NumReach/SAT outperforms all the heuristic planners in a basic domain with tightly constrained resources. This is an important result for the planning community, suggesting to seriously reconsider numeric planning.³ We finally present initial, encouraging results on using our approach in model-checking domains.

2 Approximating Variable Domains

To spell out the algorithm building the NRPG, we need some notations. I , V , and A respectively denote the initial state, set of numeric variables, and action set of the planning task. The initial state is a set of propositions – those that are initially true; also, it assigns a value to every variable $v \in V$. A numeric constraint has the form $\mathcal{P} \ 1[X] \bowtie \mathcal{P} \ 2[Y]$ where $\bowtie \in \{<, \leq, =, \geq, >, \neq\}$, and $\mathcal{P} \ 1, \mathcal{P} \ 2$ are expressions made of constants, operators in $\{+, -, *, /\}$, and variables $X \subseteq V$ respectively $Y \subseteq V$. Each action $a \in A$ consists of its precondition $\mathcal{P} \ a$, and a set E_a of effects. Each effect $e \in E_a$ consists of its condition $cn \ e$, its adds add , and its deletes $del \ e$. Additionally e is annotated with a set of numeric effects of the form $v := \mathcal{P}$. Adds and deletes are, as usual, proposition sets. Preconditions $\mathcal{P} \ a$ and effect conditions $cn \ e$ are conjunctions of propositions and numeric constraints. The semantics are defined in the obvious standard way.

1. $t := 0, P_0 := I$, for all $v \in V: D_0(v) := \{I(v)\}$
2. **while** $t < b$ **do**
3. $A_t := \{a \mid a \in A, (P_t, \overline{D}_t|_X) \models \mathcal{P} \ a[X]\}$
4. $E_t := \{e \mid \exists a \in A_t : e \in E_a, (P_t, \overline{D}_t|_X) \models cn \ e[X]\}$
5. $P_{t+1} := P_t \cup \{p \mid \exists e \in E_t : p \in add \ e\}$
6. for all $v \in V: D_{t+1}(v) := D_t(v) \cup \{c \mid$
7. $\exists a \in A_t, e \in E_a \cap E_t, (v := \mathcal{P} \ [X]) \in e, \bar{c} \in \overline{D}_t|_{X \cup Y} :$
8. $c = \mathcal{P} \ (\bar{c}|_X), (P_t, \bar{c}|_Y) \models (\mathcal{P} \ (a) \wedge cn \ (e))[Y]\}$
9. **endwhile**

Figure 1: The numeric relaxed planning graph (NRPG) algorithm, approximating reachable variable domains.

Pseudo code for the NRPG is shown in Figure 1. The algorithm keeps track of sets P_t of reached propositions, and of sets $D_t(x)$ of reached values (the reached variable domains), at time steps t . Note the slight abuse of notation for I in line

²Implementing and evaluating a planner using method (A) is future work.

³The heuristic planners employ very greedy searches, which do not work well when resources are tight. More in Section 4.

1. By b (line 2), we denote the plan length bound we are currently considering. By writing $\phi[Y]$ for a conjunction ϕ of propositions and numeric constraints (lines 3, 4, 8), we mean that $Y \subseteq X$ is the set of all variables mentioned by the numeric constraints in ϕ . By \overline{D}_t (lines 3, 4, 7), we denote the cross-product of the $D_t(v)$ for $v \in V$, i.e., $\overline{D}_t := D_t(v_1) \times \dots \times D_t(v_n)$ where $V = \{v_1, \dots, v_n\}$. For a set \overline{C} of value vectors, or for a single value vector \bar{c} , by $\overline{C}|_X$ respectively $\bar{c}|_X$ (lines 3, 4, 7, 8) we denote the vector (set) restricted to the variables X . For a set P of propositions P , a set \overline{C} of value vectors, and a conjunction $\phi[X]$ of propositions and numeric constraints, we say that $(P, \overline{C}|_X) \models \phi$ (lines 3, 4, 8) iff ϕ 's propositions are a subset of P , and there exists $\bar{c} \in \overline{C}$ so that $\bar{c}|_X$ satisfies ϕ 's numeric constraints. By $\mathcal{P} \ (\bar{c})$ (line 8) for an expression and value vector, we mean the outcome of inserting the values into the expression.

In words, the NRPG is built as follows. The reached propositions and variable domains are first set to the initial values. Then one iterates over time steps t until the desired bound is reached. In each step, the action set A_t is the set of all actions whose precondition can be satisfied at t . The same is done for the effect set E_t . The propositions reached at $t + 1$ are those that are either reached at t , or added by an effect at t . The reached variable values at $t + 1$ are those that are either reached at t , or that are the result of an effect at t and $a \ u \ l \ a \ e \ t \ a \ t \ t \ a \ t \ a \ i \ f \ i \ s \ t \ e \ c \ n \ j \ u \ c \ i \ n \ f \ i \ h \ e \ p \ a \ d \ i \ t \ a \ n \ d \ c \ a \ d \ i \ t \ a \n \ f \ i \ h \ e \ p \ i \ v \ a \ c \ i \ n \ a \n \ d \ f \ f \ e \ t$ – intuitively, this means that the insertion of new values is “guarded” by the corresponding conditions.

Example 1. Consider a task with a single variable v , which is initially 4, and a single action whose only effect has the condition $v \geq 2$ and the effect $v := v - 2$. We will have $D_0(v) = \{4\}$, $D_1(v) = \{4, 2\}$, and $D_2(v) = \{4, 2, 0\}$. Then the algorithm will stop – since the effect $v := v - 2$ is guarded by the condition $v \geq 2$, the value $v = -2$ is *not* inserted into $D_3(v)$.

The NRPG is admissible, i.e., $\mathcal{P} \ a \ l \ a \ e \ c \ h \ a \ b \ l \ e \ f \ o$ $a \ u \ i \ a \ b \ l \ e \ v \ i \ n \ t \ p \ i \ s \ g \ u \ a \ n \ d \ t \ b \ e \ c \ a \ i \ n \ d \ i \n$ $D_t(v)$. However, the algorithm is exponential in the arity – the number of variables mentioned by – conjunctions of preconditions, effect conditions, and effect expressions: if the maximum such arity is k , and the largest involved variable domain has size m , then the number of vectors to be considered is bounded only by m^k . Hoffmann [2003] avoids this blow-up by keeping only the *max* and *min* value of each $D_t(v)$, and accordingly approximating the satisfaction of (linear) numeric constraints. Kupferschmid et al [2006] largely avoid the blow-up by treating every numeric constraint in separate rather than addressing conjunctions, thus potentially decreasing the above k a lot.⁴ Our rationale here is that: 1. In difference to these works we need to compute the NRPG only once, rather than in every state of a forward search. 2. The arity of numeric constraints, and even conjunctions of them, tends to be small in many planning domains – e.g., expressing constraints on resource availability usually involves only

⁴Another difference to both Hoffmann [2003]'s and Kupferschmid et al [2006]'s algorithms lies in the “guarding” of value insertions as described above: this is a new technique.

a single variable. Indeed, in our experiments we did not find a single domain where this potential blow-up was a problem.

Another potential blow-up turned out to be more severe: the size of the domain of a variable v grows exponentially in t if every sequence of actions results in a different value for v . This happens in one of our test domains, where v is a resource, and every action using the resource has a random float cost. A remedy to this would be to give up on admissibility of the NRPG (and, with that, give up plan optimality), and try greedy approximations of variable domains instead. One could, for example, build a non-numeric planning graph first and then build the variable domains only for a seemingly relevant subset of the actions (e.g., a relaxed plan). We leave this topic open for future work. Herein we show that, in 5 out of 6 numeric domains from the international planning competition, the problem does not occur: if the granularity of numeric effects is less than that of random floats, many action sequences result in the same value.

We finally remark that one could also insert more intelligence into the NRPG, to prune non-reachable variable values. For example, an extension of Graphplan mutex reasoning [Blum and Furst, 1997] might turn out to be beneficial in some domains. We leave this open for future work.

3 SAT and SMT Encodings

We first describe the SMT encoding underlying NumReach/SMT, since that encoding is simpler. We then explain how to obtain the SAT encoding underlying NumReach/SAT. The SMT encoding makes use of decision variables for propositions, numeric variables, actions, and effects, at time steps. Precisely, there are the decision variables: p_t for all $0 \leq t \leq b$ and $p \in P$; v_t for all $0 \leq t \leq b$ and $v \in V$; a_t for all $0 \leq t < b$ and $a \in A$; and e_t for all $0 \leq t < b$ and $e \in E$. All variables except the x_t are Boolean. There are the following clauses (illustrated with Example 1):

- **Initial values** Specify the initial values of propositions and numeric variables. For example, $\{v_0 = 4\}$.
- **Actions** An action at t implies its preconditions at t . An effect at t implies its conditions at t . For example, $\{-e_t, v_t \geq 2\}$.
- **Action and effect** An effect at t implies the corresponding action at t . An action at t implies, for each effect e , that either e occurs or one of e 's conditions is false. For example, $\{-a_t, \neg v_t \geq 2, e_t\}$.
- **Effect change** An effect at t implies its adds and (negated) deletes at $t + 1$, and implies its numeric effects between t and $t + 1$. For example, $\{-e_t, v_{t+1} = v_t - 2\}$.
- **Frame axioms** Any proposition that is true at t stays true at $t + 1$ unless it is deleted. Any proposition that is false at t stays false at $t + 1$ unless it is added. Any numeric variable has the same value at t and $t + 1$ unless it is affected. For example, $\{v_{t+1} = v_t, e_t\}$.
- **Goal**: the goal condition is true at time b .
- **Mutex** Interfering actions and effects do not occur in the same step. Actions/effects interfere if they harm each other's preconditions/conditions/adds, or if they affect the same numeric variable.

Note here that this specification makes use of the NRPG, in the definition of the set of decision variables. This corresponds to our current implementation. However, *incere*

clad a il ydfinea n Mlacdi ng tu t dantate
~~*te tent nestbul di ng hest i snvi ncl dd*~~
~~*i nttahR cl/lti noi n poms*~~

For NumReach/SAT, the NRPG is essential (and its runtime is, of course, included). The encoding is identical to the above, except for the treatment of numeric variables. Precisely, instead of the v_t variables, we now have one separate (Boolean) decision variable $(v = c)_t$ for all t, v , and $c \in D_t(v)$. We further have, for all t , one decision variable $c_{\#}_t$ for every numeric constraint (every different $c_{\#} = (p \ 1 \bowtie \ p \ 2)$) occurring in the task. Finally we have, for all t , one decision variable $c_{\#}_t - \bar{c}_t$ for every $c_{\#} [X]$ and every value tuple $\bar{c} \in \bar{D}_t[X]$ that satisfies $c_{\#}$. The intended behavior is that $c_{\#}_t - \bar{c}_t$ is true iff the values it specifies are true, and that $c_{\#}_t$ is true iff one of its value tuples is true. We ensure this behavior with the following additional clauses:

- **Disjunction** For $c_{\#} [X]$, $c_{\#}_t$ is equivalent to the disjunction of all $c_{\#}_t - \bar{c}_t$ where $\bar{c} \in \bar{D}_t[X]$ satisfies $c_{\#}$.
- **Conjunction** For $c_{\#} [X]$, each $c_{\#}_t - \bar{c}_t$ is equivalent to the conjunction of $(x_1 = c_1)_t, \dots, (x_k = c_k)_t$, where $X = \{x_1, \dots, x_k\}$ and $\bar{c} = (c_1, \dots, c_k)$.

With this, the “conditions” clauses and the “actions and effects” clauses translate effortlessly, since we have a decision variable for every numeric constraint occurring in a precondition or an effect condition. We can use these variables just like the variables for propositions. Similarly, the frame clauses for numeric values now look almost like those for propositions, for example $\{e_t, \neg(v = 4)_t, (v = 4)_{t+1}\}$ – if e is not applied, and $v = 4$ at t , then $v = 4$ at $t + 1$. The only aspect that becomes somewhat complicated are the “effect state change” clauses for numeric variables. What we say is, for every numeric effect $v := p [X]$ of effect $e \in E(a)$, and for every tuple $(x_1 = c_1, \dots, x_k = c_k)$ in X that complies with at least one tuple in \bar{D}_t satisfying $p(a) \wedge c_{\#}(e)$, either e is not applied, or one of $(x_i = c_i)_t$ is false, or $(x = p(\bar{c}))_{t+1}$ is true. For example, $\{-e_t, \neg(v = 4)_t, (v = 2)_{t+1}\}$.

4 Experimental Results

The experiments were run on a cluster of machines, each with 2GByte main memory, running two Pentium III processors at 1GHz. We applied a memory cutoff of 1GByte in each run. Time-out was either 10000 seconds or 1000 seconds. NumReach/SAT and NumReach/SMT are implemented in C. We compare to Metric-FF, LPG, and SGPlan. We do not compare to the SMT-based planners LPSAT and TM-LPSAT. LP-SAT is outdated; in particular, it does not parse PDDL and we would have to translate all our instances into its input format. We do not compare to TM-LPSAT since, though contacting the authors, we did not manage to obtain an executable.

4.1 Competition Examples

We first consider numeric domains run in the international planning competition. Precisely, we consider the numeric domains run in the 2002 competition. The reason why we do not run domains from 2004 or 2006 is that, since 2002, the

Instance	SAT	SMT	FF	SGPlan	LPG
Depots 10	87.8	672.2	0.1	0.1	0.2
Depots 13	35.6	245.0	0.1	0.1	0.3
Depots 14	1083.7	–	0.5	0.5	0.7
Depots 16	30.0	470.9	0.2	0.2	0.3
Depots 17	81.1	–	1.1	1.1	0.9
Settlers 1	18.6	2042.8	8.5	0.4	6.5
Settlers 2	1.6	53.5	0.1	0.4	0.4
Settlers 3	–	–	59.3	0.5	1.3
Settlers 4	22.6	775.2	163.4	0.7	0.7
Settlers 5	1480.7	–	1.1	1.0	1.5
Settlers 6	–	–	19.2	0.8	2.3
Settlers 7	–	–	–	1.4	33.7
Zeno 8	8.5	26.0	0.0	0.0	0.0
Zeno 9	18.1	299.3	0.0	0.0	0.1
Zeno 10	126.1	1397.8	0.1	0.1	0.1
Zeno 11	85.7	120.2	0.1	0.1	0.1
Zeno 12	480.7	6585.8	0.1	0.1	0.10
Rovers 8	3.7	59.2	0.1	442.7	0.0
Rovers 9	156.9	–	–	–	0.2
Rovers 10	70.9	455.4	0.1	0.1	0.1
Rovers 11	43.0	981.4	6.6	6.3	0.3
Rovers 12	1.9	33.5	0.1	0.1	0.3
Rovers 13	503.5	–	–	1945.7	1.2
Rovers 14	20.3	402.7	–	–	0.7
Rovers 15	104.7	2860.3	–	0.1	63.3
Rovers 16	352.7	–	–	1222.5	10.8
Rovers 17	809.9	–	–	–	126.8
Rovers 18	77.9	–	–	–	14.3
Rovers 19	MEM	–	–	–	0.8
Rovers 20	MEM	–	–	–	–

Table 1: Runtime results in seconds, in four of the six 2002 competition domains (see text). Dashes mean time-out (10000 seconds), MEM out of memory (1GByte). Some domain and planner names abbreviated as obvious.

focus of the competition has shifted away from numeric planning (though as we will see that is far from being “solved”). The 2002 numeric domains are thus still the main benchmark for this context. They are named Depots, Driverlog, Rovers, Satellite, Settlers, and Zenotravel. Depots is a combination of logistics and blocksworld (with, e.g., numeric restrictions on the load status of trucks). Driverlog has trucks that need drivers; the numeric variables (distances walked and driven) are mentioned only in the optimization metric. Rovers is about gathering rock/soil samples and images, with energy usage and a recharge operator that can be applied only in “sunny spots”. Satellite is about gathering images, also with energy usage (no recharge). Zenotravel is a simple logistics-type problem with fuel usage and a refuel operator that can be applied anywhere anytime. Table 1 shows our data.

Driverlog is left out of Table 1 since all tested planners ignore the optimization metric – and with it the numeric variables – anyway. Satellite is left out of the table because in this domain our approach is trivially nonsense: any “satellite” can turn between any two “directions”; doing so costs the respective “slew time” in energy; in the competition instances, the slew times are assigned basically as $\frac{1}{n}$. Because of the latter, every sequence of turning actions has a different summed up cost, yielding a different possible numeric value. So the reachable variable domains grow exponentially in the depth of the NRPG, which exhausts memory even in the smallest instance of the competition suite. We find it doubtful whether such a scenario (random float costs) makes much sense – why would one need to distinguish between arbitrar-

ily many different action costs? To explore this topic further, in Section 4.2 we run an experiment explicitly scaling the size of the set of numbers from which action costs are drawn.

Each competition test suit contains 20 instances; every instance solved by NumReach/SMT was also solved by NumReach/SAT. For Depots and Zenotravel, Table 1 shows data for the 5 largest instances solved by NumReach/SAT. For Settlers, we show the 7 smallest instances, which contain everything solved by NumReach/SAT and Metric-FF (even the small Settlers instances are comparatively large, requiring plans with several dozen steps). For Rovers, since we consider that particularly interesting, we show all instances except the smallest ones. From a quick glance, one sees that

all the instances This underlines the merits of our approach quite impressively. For the sub-optimal planners, neither of Depots, Settlers, and Zenotravel constitutes much of a problem. The single exception is Metric-FF in Settlers, which is only marginally faster than NumReach/SAT. In Rovers, however, NumReach/SAT outperforms not only NumReach/SMT but also Metric-FF and SGPlan, being beat only by LPG.⁵ This is due to the nature of energy consumption and recharge in this domain: to recharge, one has to first reach a sunny spot. Metric-FF and SGPlan presumably spend excessive time in large dead ends where there is not even enough energy left to reach such a spot. LPG’s search has a more stochastic nature, which is less affected by this. In the next section we run an experiment explicitly scaling the constrainedness of resources.

4.2 Transport

We constructed a simple transportation domain, called Transport, to run some targeted experiments. A truck moves in a weighted graph. A set of packages must be transported between nodes. Actions move along edges, and load/unload packages. Each edge costs its weight in fuel. There is *no* refuelling, so once the truck ran out of fuel it’s “game over”.⁶ While this domain is simplistic, we believe it serves well as an abstract model of the relevant issues in planning with resources. In particular, we can scale the instances on variable domain size and on resource constrainedness, as follows. Our instance generator takes the parameters N , P , M , and C . First a random connected (undirected) graph with N nodes is created. Then P packages are added, with random origins and destinations. The edge weights are uniformly drawn from the set $\{1, 2, \dots, M\}$. Then a domain-specific branch-and-bound procedure, which we developed especially for this purpose, computes the minimum amount of fuel, min_fuel , needed to solve the task. The initial fuel supply of the truck is set to $\lfloor C * min_fuel \rfloor$. This way, the size of the reachable variable domains can be controlled by M , and the resource constrainedness can be controlled by how close C is to 1.0.

⁵In instances 19 and 20, NumReach/SAT’s CNFs become too large to fit into memory (see also the next section).

⁶We consider a single truck rather than multiple trucks for the sake of simplicity. Note that this puts our own SAT-based approach at a disadvantage vs. the heuristic planners: SAT-based planners are well known to work much better when there is a lot of parallelism, reducing the number of time steps that must be considered.

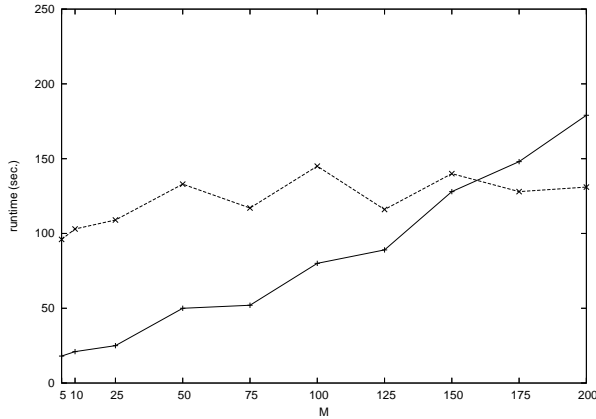


Figure 2: Scaling variable domain size.

In our first experiment, we aim to see how variable domain size affects the planners, so we scale M . We keep N and P fixed to 7, which is challenging but feasible. We fix $C = 1.1$, which is relevant since, as we shall see, in this region the SAT-based approach is more efficient than all state-of-the-art heuristic planners. Runtime plots for NumReach/SAT and NumReach/SMT are in Figure 2; each data point is the mean value of 100 instances. We see that NumReach/SAT has a strong advantage with low M ; as expected it gets worse with growing M , while NumReach/SMT reacts only very slightly to the value of M .

We ran another experiment where we scaled M further, over $M = 300, \dots, 1000$. As expected, NumReach/SMT's behavior does not change much, while NumReach/SAT degrades further. With a time-out of 1000 seconds, NumReach/SAT's solution percentage as we increase M in steps of 100 is 94, 81, 50, 48, 30, 23, 18, 10. We remark that the limiting factor here is neither NRPG building nor CNF building nor even SAT reasoning: the *size* of the CNFs is what makes the solution rate drop. As a rule of thumb, until around 6 million clauses NumReach/SAT beats NumReach/SMT, between 6 and 10 million clauses NumReach/SMT beats NumReach/SAT, and after that, which in our experiment here happens starting with $M = 300$, the propositional CNFs don't fit into 1GByte memory any longer. Note that this provides a simple rule to automatically choose between NumReach/SAT and NumReach/SMT: just compute the number of clauses in the propositional CNF, and apply a threshold.

We emphasize that in the competition domains, Table 1, the advantage of NumReach/SAT over NumReach/SMT is much larger than even for $M = 5$ in Figure 2. Presumably, NumReach/SMT has more trouble than NumReach/SAT with the mix of planning aspects in these domains – remember that Transport is very basic.

The experiment shown in Figure 3 scales C between 1.0 and 2.0. N and P are fixed to 8, M is fixed to 25. The left hand side y axis is *log scale* runtime. Each data point is the average over 100 instances, where time-out is 10000 seconds, which is inserted into the average computation in unsolved cases. To complement this, the right hand side y axis shows the percentage of solved instances. NumReach/SAT's curve is the horizontal line; all instances are solved, so no solution percentage is shown. Note that the seemingly linear nature of

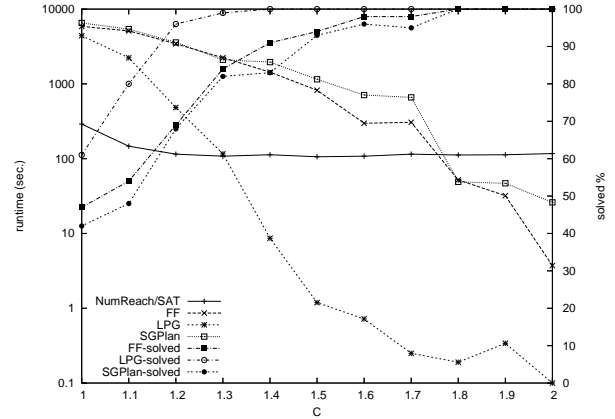


Figure 3: Scaling resource constrainedness.

the heuristic planners' runtime curves, as we move towards $C = 1.0$, is just an artefact of the runtime cutoff. The sharp drops in solution percentage more reliably reveal the main message of Figure 3: *the heuristic planners' efficiency is not a constant, it fails as resource constrainedness increases*. Ours is the first experiment making this important point absolutely clear. Even though NumReach/SAT is optimal, it outperforms the heuristic planners. This inverts (almost) every result of the planning competitions since 2000. Note that Transport is not an awkward artificial problem, but a natural and relevant application of planning.

4.3 Jugs-and-Water

In the Jugs-and-Water domain, one has a set of jugs of varying sizes, and one wants to achieve a given fill status for all the jugs. Actions fill or empty a jug, or pour the content of one jug into another one. Thus, in this domain, numeric variables (fill status of jugs) are not resources but constitute highly interacting phenomena, perhaps similar to Rubic's Cube or so, from the "point of view" of a general purpose solver. We obtained random instances with N jugs and M maximum jug size by choosing jug size uniformly from $\{1, 2, \dots, M\}$, and then using a repeated greatest common divisor computation to obtain solvable goals.

In some way, M here corresponds to the M parameter of transport: it scales the possible variable value ranges. However, very much unlike Transport, in Jugs-and-Water M has a huge effect on instance hardness. The larger M is, the longer will the plan typically be, making it much harder to find. Table 2 shows results scaling N and M simultaneously. For readability, we only show the solution rates. We could not run LPG since that cannot handle conditional effects (of the action pouring the content of one jug into another). NumReach/SAT vastly outperforms NumReach/SMT. The same goes for Metric-FF, giving another example of the unusual dominance of an optimal over a sub-optimal planner. It is unclear to us what the reason for SGPlan's efficiency is.

4.4 Model Checking

To point out that our new method might make sense in some cases of model checking as well, we have run a few preliminary experiments. We modelled some model checking toy examples in PDDL. Many of those examples make use of

$N \setminus M$	4	8	12	16	20
NumReach/SAT					
3	100	100	98	100	96
6	100	100	100	100	92
9	100	100	100	100	78
12	100	98	96	94	62
15	96	96	78	66	62
NumReach/SMT					
3	98	92	86	78	62
6	98	92	92	82	72
9	96	92	70	62	42
12	86	58	28	20	8
15	60	44	12	10	2
Metric-FF					
3	100	100	100	100	100
6	100	100	100	100	100
9	100	96	60	60	54
12	100	82	34	34	24
15	100	70	12	12	6
SGPlan					
3	100	100	100	100	100
6	100	100	100	100	100
9	100	100	100	100	98
12	100	100	98	100	92
15	100	100	100	96	92

Table 2: Solution rate for the Jugs-and-Water domain. 100 instances per data point, time-out 1000 seconds.

arrays, with arithmetics on the array indices. This is very awkward to model in PDDL, without an explicit notion of arrays. After some exploration of the relevant literature, we ended up using variants of the “Fischer Protocol for Mutual Exclusion” and the “Bakery Protocol for Mutual Exclusion” (the latter has an array, but no arithmetics on the indices). In both cases, there are N processes, at most one of which should be in a “critical region” at any point in time. In both cases, versions with bugs are mentioned in the literature; these versions we mapped into PDDL.⁷ Planning in these domains is about finding sequences of transitions ending in a state where at least two processes are in the critical region. For the “Fischer” domain, we obtained very good scaling. With $N = 10, 20, 30, 40, 50, 100$ processes, NumReach/SAT | NumReach/SMT runtime is 0.2|1.8, 0.4|6.4, 0.7|12.4, 2.5|20.9, 1.9|33.6, 13.4|118.4, showing a consistent advantage for NumReach/SAT. In the Bakery domain, we obtained much worse scaling. For NumReach/SAT, this is due to an awkward encoding we needed for a *max* operation, making the NRPG grow rather quickly. For NumReach/SMT, excessive time is taken in satisfiability testing. Concretely, for $N = 2, 3, \dots, 10$ the data we get is the following, showing SAT time for NumReach/SAT | NRPG time for NumReach/SAT | SMT time for NumReach/SMT: 0.0|0.0|0.2, 0.0|0.0|0.5, 0.0|0.0|1.0, 0.0|0.0|2.2, 0.0|0.2|6.5, 0.1|0.9|15.8, 0.1|4.3|32.5, 0.2|47.3| > 1000. The advantage for NumReach/SAT in terms of SAT solving time is dramatic.

5 Conclusions and Further Directions

We have developed a third technique for the application of satisfiability testing to solving state-space reachability prob-

⁷The “Fischer” domain is originally formulated for Timed Automata, with continuous-valued clocks. Since we cannot handle continuous updates, we made a straightforward modification using just integers with discrete updates (“counting” time units).

lems in numeric domains. Our empirical results suggest that, at least in planning, the technique is very competitive.

We recommend to the planning community to reconsider numeric planning. Our results in resource-constrained transportation indicate that the relevant issues there may be quite different from what we have got used to elsewhere.

In the future, we intend to explore the two lines of research modifying our current NRPG technique, outlined in Section 2: 1. Try to make the NRPG more intelligent by additional reasoning. 2. Try to avoid any blow-ups altogether by a more greedy approximation of variable domains. We also consider it interesting to explore how method (A) – encoding numbers as bitvectors – will work in numeric planning.

References

- [Blum and Furst, 1997] A. Blum and M. Furst. Fast planning through planning graph analysis. *AI* 90(1-2):279–298, 1997.
- [Bozzano *et al.*, 2006] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. A tight integration of SAT and mathematical decision procedures. *JAR* 2006.
- [Chen *et al.*, 2006] Y. Chen, B. Wah, and C. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *JAR*, 2006.
- [Clarke *et al.*, 2001] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *FMD*, 19(1):7–34, 2001.
- [Clarke *et al.*, 2004] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *CADE*, pages 168–176, 2004.
- [Een and Sörensson, 2003] N. Een and N. Sörensson. An extensible SAT solver. In *CADE*, 2003.
- [Ganzinger *et al.*, 2004] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CADE*, pages 175–188, 2004.
- [Gerevini *et al.*, 2003] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *JAR*, 20:239–290, 2003.
- [Hoffmann, 2003] J. Hoffmann. The Metric-FF planning system: Translating “ignoring delete lists” to numeric state variables. *JAR*, 20:291–341, 2003.
- [Kautz and Selman, 1999] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *JAIR*, pages 318–325, 1999.
- [Kupferschmid *et al.*, 2006] S. Kupferschmid, J. Hoffmann, H. Dierks, and G. Behrmann. Adapting an AI planning heuristic for directed model checking. In *CADE*, pages 35–52, 2006.
- [Seshia and Bryant, 2004] S. Seshia and R. Bryant. Deciding quantifier-free presburger formulas using parameterized solution bounds. In *CADE*, 2004.
- [Shin and Davis, 2005] J. Shin and E. Davis. Processes and continuous change in a SAT-based planner. *AI* 166:194–253, 2005.
- [Wolfman and Weld, 1999] S. Wolfman and D. Weld. The LPSAT engine & its application to resource planning. In *JAIR*, pages 310–316, 1999.