

A Novel Transition Based Encoding Scheme for Planning as Satisfiability

Ruoyun Huang, Yixin Chen, Weixiong Zhang

Dept. of Computer Science and Engineering
Washington University in St. Louis
Saint Louis, MO, 63105, USA
{rh11, chen, zhang}@cse.wustl.edu

Abstract

Planning as satisfiability is a principal approach to planning with many eminent advantages. The existing planning as satisfiability techniques usually use encodings compiled from the STRIPS formalism. We introduce a novel SAT encoding scheme based on the SAS+ formalism. It exploits the structural information in the SAS+ formalism, resulting in more compact SAT instances and reducing the number of clauses by up to 50 fold. Our results show that this encoding scheme improves upon the STRIPS-based encoding, in terms of both time and memory efficiency.

Introduction

Planning as satisfiability is one of the major paradigms for planning. The approaches using this technique compile a planning problem into a sequence of SAT instances, with increasing time horizons (Kautz, Selman, and Hoffmann 1999). Planning as satisfiability has a number of distinct characteristics that make it effective and widely applicable. It makes a good use of the extensive advancement in fast SAT solvers. The SAT representation can be extended to accommodate a variety of complex problems, such as planning with uncertainty (Castellini, Giunchiglia, and Tacchella 2003), numerical planning (Hoffmann et al. 2007) and temporally expressive planning (Huang, Chen, and Zhang 2009). Developing novel and superior SAT encoding schemes has a great potential to advance the state-of-the-art of planning.

Encoding scheme has a great impact on the efficacy of SAT-based planning. Extensive researches have been done on compact SAT encoding for planning. One example of compact encoding is the lifted action representation, first studied in (Ernst, Millstein, and Weld 1997). In this compact encoding scheme, actions are represented by a conjunction of parameters, thus this method mitigates the problem of blowing up time steps caused by grounding and itemizing each action. The original scheme does not guarantee the optimality on time steps. An improved lifted action representation that preserves optimality was proposed in (Robinson et al. 2009). In (Rintanen, Heljanko, and Niemelä 2006), a

new encoding was proposed based on a relaxed parallelism semantic, which still does not guarantee optimality.

Most planning approaches heavily depend on problem formulations. The SAS+ formalism represents a planning problem using multi-valued state variables instead of propositional facts (Bäckström and Nebel 1996). The SAS+ formalism has recently attracted much attention due to its concise representation and rich structural information (Helmert 2006). It has been used to derive heuristics (Helmert 2006), landmarks (Richter, Helmert, and Westphal 2008), and stronger mutual exclusions (Chen et al. 2009).

In this paper, we propose the first SAS+ formalism-based SAT encoding scheme for classical planning. Unlike previous SAT encoding methods that model STRIPS actions and facts, the new SAT encoding directly models *transitions* in the SAS+ formalism.

We further study the search space in our new SAT encoding. We show that the new encoding scheme models a planning problem with a search space consisting of two hierarchical subspaces. The top subspace is the space of *transition plans*, and the lower subspace is the space of *supporting action plans* corresponding to feasible transition plans. We analyze and compare the worst case search space sizes, in the original STRIPS-based search space and the new hierarchical search space. We show that, because of the new search space's structure, in which the lower level search can be decomposed based on time steps without backtracking, our encoding scheme typically has a smaller search space.

We also propose a number of techniques to reduce the encoding size by recognizing the characteristics of mutual exclusive cliques of actions and transitions. Finally, we evaluate the new encoding on the standard benchmarks from recent planning competitions. Our results show that the new encoding scheme only requires as low as 6% of the memory needed by a STRIPS-based encoding, is more efficient than the latter, and solves many large instances that the state-of-the-art STRIPS-based SAT planners fail to solve.

This paper is organized as follows. After giving some basic definitions in Section 2, we present our SAS+ based encoding in Section 3. We then explain the search space size reduction by the new encoding in Section 4. Techniques to further reduce the encoding size are presented in Section 5. We present our experimental results in Section 6 and conclude in Section 7.

Background

The SAS+ formalism (Bäckström and Nebel 1996) represents a classical planning problem by a set of multi-valued **state variables**. A planning task Π in the SAS+ formalism, referred to as a SAS+ planning problem, is defined as a tuple $\Pi = \{\mathcal{X}, \mathcal{O}, s_{\mathcal{I}}, s_{\mathcal{G}}\}$, where

- $\mathcal{X} = \{x_1, \dots, x_N\}$ is a set of *state variables*, each with an associated finite domain $Dom(x_i)$;
- \mathcal{O} is a set of actions and each action $a \in \mathcal{O}$ is a tuple $(pre(a), eff(a))$, where $pre(a)$ and $eff(a)$ are sets of partial state variable assignments in the form of $x_i = v, v \in Dom(x_i)$;
- A **state** s is a set of assignments with all the state variables assigned. We denote \mathcal{S} as the set of all states, $s_{\mathcal{I}} \in \mathcal{S}$ the initial state, and $s_{\mathcal{G}}$ a partial assignment that defines the goal. A state $s \in \mathcal{S}$ is a goal state if $s_{\mathcal{G}} \subseteq s$.

Given a state s , in which a state variable x is assigned a value $f \in Dom(x)$, we write $s(x) = f$ for this assignment. For a given state s and an action a , when all variables in $pre(a)$ matches the assignments in s , a is called *applicable* in state s . We use $apply(s, a)$ to denote the state after applying a to s , in which variable assignments are changed according to $eff(a)$.

We also write $apply(s, P)$ to denote the state after applying a set of actions $P, P \subseteq \mathcal{O}$, to s . A set of actions P is applicable to s , when 1) each $a \in P$ is applicable to s , 2) there are no two actions $a_1, a_2 \in P$ existing such that a_1 and a_2 are mutually exclusive (Blum and Furst 1997).

Definition 1 (Action Plan). An action plan is a sequence $P = \{P_1, P_2, \dots, P_N\}$, where each $P_t, t \in \{1, 2, \dots, N\}$, is a set of actions executed at time step t , such that $s_{\mathcal{G}} \subseteq apply(\dots apply(apply(s_{\mathcal{I}}, P_1), P_2) \dots P_N)$.

In this paper, we also consider plans of transitions, based on the SAS+ formalism.

Definition 2 (Transition). Given a state variable x , a transition is a change of the assignment of x from value f to g , written as $\delta_{f \rightarrow g}^x$, or from an unknown value to g , written as $\delta_{* \rightarrow g}^x$. We may also simplify the notation of $\delta_{f \rightarrow g}^x$ to δ^x or δ , when there is no confusion.

For an action a , three types of transitions may be derived:

- Transitions $\delta_{f \rightarrow g}^x$ such that $(x = f) \in pre(a)$, and $(x = g) \in eff(a)$. A transition $\delta_{f \rightarrow g}^x$ is applicable to a state s , if and only if $s(x) = f$. We denote $apply(s, \delta)$ as the state after applying transition δ to state s , which results in a new state s' such that $s'(x) = g$.
- Transitions $\delta_{f \rightarrow f}^x$ such that no assignment to x is in $eff(a)$, and $(x = f) \in pre(a)$. We call this type of transitions **prevailing**. A prevailing transition $\delta_{f \rightarrow f}^x$ is applicable to a state s , if and only if $s(x) = f$, resulting in the same state s .
- Transitions $\delta_{* \rightarrow g}^x$ such that no assignment to x is in $pre(a)$, and $(x = g) \in eff(a)$. We call this type of transitions **mechanical**. A mechanical transition $\delta_{* \rightarrow g}^x$ can be applied to an arbitrary state s , and the result of $apply(s, \delta_{* \rightarrow g}^x)$ is a state s' with $s'(x) = g$.

For each action a , we denote its **transition set** as $M(a)$, which includes all three types of transitions above. A transition is applicable at a state s only for the above cases. Given a transition δ , we use $A(\delta)$ to denote the set of actions that include δ in their transition sets. We call $A(\delta)$ the **supporting action set** of δ .

We use $R(x) = \{\delta_{f \rightarrow f}^x \mid \forall f, f \in Dom(x)\}$ to denote the set of all prevailing transitions related to x , and R the union of $R(x)$ for all $x \in \mathcal{X}$. Note that for a prevailing transition $\delta \in R$, there may not be an action a such that $\delta \in M(a)$. We introduce $\mathcal{T}(x) = \{\delta^x \mid \exists a \in \mathcal{O}, \delta^x \in M(a)\} \cup R(x)$, which is the set of all legal transitions that transit the state variable x between two assignments. \mathcal{T} is the union of $\mathcal{T}(x)$, for all $x \in \mathcal{X}$.

Definition 3 (Transition Mutex). Two different transitions δ_1 and δ_2 are mutually exclusive, i.e., δ_1 and δ_2 are a pair of transition mutex, if there exists a state variable $x \in \mathcal{X}$ such that $\delta_1, \delta_2 \in \mathcal{T}(x)$, and either of the following holds:

1. Neither δ_1 nor δ_2 is mechanical transition.
2. Both δ_1 and δ_2 are mechanical transitions.
3. Only one of δ_1 and δ_2 is a mechanical transition, and without loss of generality, they are of the form $\delta_{f \rightarrow g}^x$ and $\delta_{* \rightarrow g}^x$, respectively.

Two transitions that are both mechanical or not, are mutual exclusive to another, as long as they belong to the same state variable. If exactly one of them is mechanical, then they are mutual exclusive if and only if they transit to the same assignment.

A set of transitions T is applicable to a state s when 1) every transition $\delta \in T$ is applicable to s , and 2) there do not exist two transitions $\delta_1, \delta_2 \in T$ such that δ_1 and δ_2 are mutually exclusive. We write $apply(s, T)$ to denote the state after applying all transitions in T to s with an arbitrary order.

Definition 4 (Transition Plan). A transition plan is a sequence $Q = \{Q_1, Q_2, \dots, Q_N\}$, where $Q_t, t \in [1, N]$, is a set of transitions executed at time step t , such that $s_{\mathcal{G}} \subseteq apply(\dots apply(apply(s_{\mathcal{I}}, Q_1), Q_2) \dots Q_N)$.

There always exists a unique transition plan for a parallel action plan. In contrast, given a transition plan, there may exist either no or multiple valid action plans.

SAS+ Based SAT Encoding

In this section, we propose our new SAS+ based encoding (SASE) for SAS+ planning tasks. Similar to the STRIPS based encoding in SatPlan, we use an increasing number of time steps, N . For a fixed N , we encode a planning task in the SAT representation which can be solved by a SAT solver. The high-level planner will start with a small N and increase N until a satisfiable solution is found. For a given N , SASE includes the following binary variables:

1. $U_{\delta, t}, \forall \delta \in \mathcal{T}$ and $t \in [1, N]$. $U_{\delta, t}$ may also be written as $U_{x, f, g, t}$ when δ is explicitly defined as $\delta_{f \rightarrow g}^x$,
2. $V_{a, t}, \forall a \in \mathcal{O}$ and $t \in [1, N]$.

We have two classes of variables, transition variables U and action variables V . We have eight classes of clauses for a planning task. In the following, we define each class for every time step $t \in [1, N]$ unless otherwise indicated.

1. Initial state - $(\forall x, s_{\mathcal{I}}(x) = f): \bigvee_{\forall \delta_{f \rightarrow g} \in \mathcal{T}(x)} U_{x,f,g,1}$.
2. Goal - $(\forall x, s_{\mathcal{G}}(x) = g): \bigvee_{\forall \delta_{f \rightarrow g} \in \mathcal{T}(x)} U_{x,f,g,N}$.
3. Transition's progression - $(\forall \delta_{f \rightarrow g}^x \in \mathcal{T} \text{ and } t \in [2, N]):$

$$U_{x,f,g,t} \rightarrow \bigvee_{\forall \delta_{f' \rightarrow f}^x \in \mathcal{T}(x)} U_{x,f',f,t-1}.$$

4. Transition mutex - $(\forall \delta_1 \forall \delta_2 \text{ such that } \delta_1 \text{ and } \delta_2 \text{ are transition mutex}): U_{\delta_1,t} \rightarrow \neg U_{\delta_2,t}$.
5. Existence of transitions - $(\forall x \in \mathcal{X}): \bigvee_{\forall \delta \in \mathcal{T}(x)} U_{\delta,t}$.
6. Composition of actions - $(\forall a \in \mathcal{O}):$

$$V_{a,t} \rightarrow \bigwedge_{\forall \delta \in M(a)} U_{\delta,t}.$$

7. Action's existence - $(\forall \delta \in \mathcal{T} \setminus R):$

$$U_{\delta,t} \rightarrow \bigvee_{\forall a, \delta \in M(a)} V_{a,t}.$$

8. Non-interference of actions - $(\forall a_1 \forall a_2 \text{ such that } \exists \delta, \delta \in T(a_1) \cap T(a_2) \text{ and } \delta \notin R): V_{a_1,t} \rightarrow \neg V_{a_2,t}$.

Clauses in class 3 specify and restrict how transitions change over time. Clauses in classes 4 and 5 enforce that exactly one related transition can be true for each state variable. Clauses in classes 6 and 7 together encode how actions are composed by transitions and how transitions imply a disjunction of actions. Clauses in class 8 restrict mutually exclusive actions from interfering with each other.

On Search Spaces of Encodings

It is in general difficult to accurately estimate the time that a SAT solver needs to solve a SAT instance, as it depends on not only the problem size, but also the structure of the clauses. In this section, we give a preliminary analysis of the worst case search space of a planning problem encoded in SASE for a given time step N . In particular, we examine the search spaces corresponding to the SAT instances in SatPlan and SASE, respectively. Our argument explains the underlying encoding structures in SASE and why a problem in SASE can be typically efficiently solved.

Search Space of STRIPS-based Encodings

We first consider the search space of planning in a STRIPS-based SAT encoding. To simplify the analysis, we focus on an action-based encoding. The argument can be readily extended to an encoding with both actions and facts explicitly represented. In an action-based encoding, one binary variable is introduced for each action a at a time step t . The constraint propagation is achieved through application of actions in this encoding; hence, a key problem is to select a set of actions for each time step t . There are

$2^{|\mathcal{O}|}$ possible subsets of actions at each time step. Therefore, a total of $(2^{|\mathcal{O}|})^N$ possible action plans in the search space. An exhaustive search will explore a search space of size $O((2^{|\mathcal{O}|})^N)$.

Search Space of SASE

A major difference between SASE and an action-based encoding is that in the former encoding, actions are not responsible for the constraint propagations over time horizons. Figure 1 illustrates their essential differences. In SASE, the SAT instance can be reduced to the following search problem of two hierarchies.

- On the top level, we search for a transition plan as defined in Definition 4. This amounts to finding a set of transitions for each time step t (corresponding to all δ such that $U_{\delta,t}$ is set to 1), so that they satisfy the clauses in classes 1-5.
- On the lower level, we find an action plan that satisfies the transition plan. In other words, for a given transition plan that satisfies clauses in classes 1-5, we find an action plan satisfying clauses in classes 6-8.

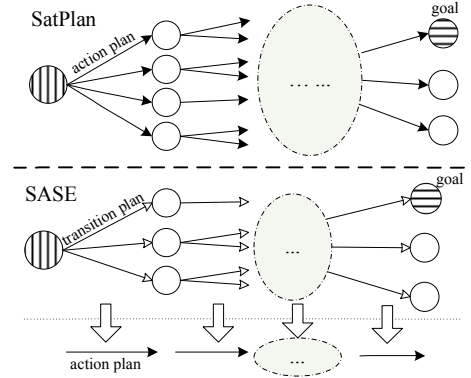


Figure 1: Illustration of how the search spaces of two encoding schemes differ from each other.

We now analyze the search space size in both hierarchies of SASE. For the top level, since there are $|\mathcal{T}|$ transitions, at each time step we have $2^{|\mathcal{T}|}$ choices thus the size is in total $(2^{|\mathcal{T}|})^N$. We note that $|\mathcal{T}|$ is usually much less than $|\mathcal{O}|$. On the lower level, two observations can be made. For a time step t and a given subset of selected transitions (corresponding to all δ such that $U_{\delta,t}$ is set to 1), finding a subset of actions that satisfies clauses in classes 6-8 amounts to exploring a search space with size $K = \prod_{\delta \in \mathcal{T}} |A(\delta)|$ in the worst case. Given a transition plan, the problems of finding a supporting action plan at different time steps are independent of one another. That is, an action plan can be found for each time step separately without backtracking across different time steps. Hence, the total cost of the lower level is NK . Therefore, to solve an SASE instance, the search space that an exhaustive search may explore is bounded by $O((2^{|\mathcal{T}|})^N NK)$.

The number of transitions $|\mathcal{T}|$ is generally much smaller than the number of actions $|\mathcal{O}|$ in practice. For instance, in Pipesworld-30, $|\mathcal{O}|$ is 15912 and $|\mathcal{T}|$ is 3474; in TPP-30, $|\mathcal{O}|$ is 11202 and $|\mathcal{T}|$ is 1988. On the other hand, although K is exponential in $|\mathcal{T}|$, it is a relatively smaller term. Therefore,

the bound of SASE $O((2^{|\mathcal{T}|})^N NK)$ is smaller than the one for STRIPS-based encoding $O((2^{|\mathcal{O}|})^N)$.

Reducing Encoding Size of SASE

We propose several techniques to reduce the size of SASE. We first represent all mutual exclusions in SASE using a more compact clique representation. We then develop two new techniques to recognize the special structure of SASE and further reduce encoding size.

Mutual Exclusion Cliques

A key observation on SASE is that mutual exclusions naturally define cliques of transitions or actions in which at most one of them can be true. There are two types of cliques: 1) for each $x \in \mathcal{X}$, $\mathcal{T}(x)$ is a *clique of transitions* enforced by the class 4 clauses, and 2) for each transition δ that is not prevailing, $A(\delta)$ is a *clique of actions* enforced by the class 8 clauses.

To encode all mutexes within a clique of size n pairwise requires $O(n^2)$ clauses. To reduce the number of clauses used, in SASE, we use a compact representation proposed in (Rintanen 2006) which uses $O(n \log n)$ auxiliary variables and $O(n \log n)$ clauses. The basic idea is the following. Suppose that we have a clique $\{x, y, z\}$ where at most one variable is true. we introduce auxiliary variables b_0, b_1 and clauses $x \Leftrightarrow \bar{b}_0 \wedge \bar{b}_1$, $y \Leftrightarrow \bar{b}_0 \wedge b_1$ and $z \Leftrightarrow b_0 \wedge \bar{b}_1$. Note that in SatPlan’s encoding, mutual exclusions are not naturally cliques like in SASE, thus the compact clique representation cannot be efficiently applied.

Reduce Subsumed Action Cliques

We observe that there exist many action cliques that share common elements, while transition cliques do not have this property. In the following, we discuss the case where one action clique is a subset of another. Given two transitions δ_1 and δ_2 , if $A(\delta_1) \subseteq A(\delta_2)$, clique $A(\delta_1)$ is referred to being subsumed by clique $A(\delta_2)$.

Instances	Before Reduction		After Reduction	
	count	size	count	size
Pipesworld-20	2548	21.72	516	53.66
Storage-20	1449	12.46	249	60.22
Openstack-10	221	22.44	141	23.4
Airport-20	1024	6.45	604	8.49
Driverslog-15	1848	2.82	1848	2.82

Table 1: Statistics of action cliques. “count” gives the number of action cliques, and “size” is the average size of the action cliques.

In pre-processing, for each transition $\delta_1 \in \mathcal{T}$, we check if $A(\delta_1)$ is subsumed by another transition δ_2 ’s action clique. If so, we do not encode action clique $A(\delta_1)$. In the special case when $A(\delta_1) = A(\delta_2)$ for two transitions δ_1 and δ_2 , we only need to encode one of them. Note that before doing this, we verify there are no two actions, such that one’s transition set is a subset of another.

Table 1 presents the number of cliques and their average sizes, before and after reducing action cliques, on some representative problems. The reduction is significant on most problem domains, except Driverslog in which no reduction

occurred. Note that the average sizes of cliques are increased since smaller ones are subsumed and not encoded.

Reduce Action Variables

Action variables form the majority of all variables. Thus, it is important to reduce the number of action variables. To this end, we propose two methods when certain structure of a SAS+ task is observed.

Unary transition reduction. Given a transition δ such that $|\mathcal{T}(\delta)| = 1$, we say the only action a in $\mathcal{T}(\delta)$ is reducible. Since a is the only action supporting δ , they are logically equivalent. For any such action a , we remove $V_{a,t}$ and replace it by $U_{\delta,t}$, for $t = 1, \dots, N$.

Unary difference set reduction. Besides unary transition variables, it is also possible to eliminate an action variable by two or more transition variables. Given a transition δ , one observation over the actions in $A(\delta)$ is that their transition sets often differ by only one transition.

Definition 5 Given a transition $\delta \in \mathcal{T}$, let $I = \bigcap_{a \in A(\delta)} M(a)$. If for every $a \in A(\delta)$, $|M(a) \setminus I| = 1$, we call action set $A(\delta)$ a unary difference set.

Consider a transition δ_1 with $A(\delta_1) = \{a_1, a_2, \dots, a_n\}$. If action set $A(\delta_1)$ is a unary difference set, the transition sets must have the following form:

$$M(a_1) = \{\delta_1, \delta_2, \dots, \delta_k, \theta_1\}$$

$$M(a_2) = \{\delta_1, \delta_2, \dots, \delta_k, \theta_2\}$$

$$\vdots$$

$$M(a_n) = \{\delta_1, \delta_2, \dots, \delta_k, \theta_n\}$$

In this case, we eliminate the action variables for a_1, \dots, a_n by introducing the following clauses. For each $i, i = 1, \dots, n$, we replace $V_{a_i,t}$ by $U_{\delta_1,t} \wedge U_{\theta_i,t}$, for $t = 1, \dots, N$. Hence, the action variables are eliminated and represented by only two transition variables.

Instances	$ \mathcal{O} $	R ₁	R ₂	%
Zeno-15	9420	1800	7620	100.00
Pathway-15	1174	173	810	83.73
Trucks-15	3168	36	300	10.61
Openstack-10	1660	0	400	24.10
Storage-10	846	540	0	63.83

Table 2: Number of reducible actions in representative instances. Column ‘R₁’ and ‘R₂’ is the number of action variables reduced, by unary transition reduction and unary difference set reduction, respectively. Column ‘%’ is the percentage of the actions reduced by both methods.

Table 2 shows the number of reducible actions in several representative problems. In Zenotravel, all action variables can be eliminated when the two reduction methods are used. In Openstack and Storage, there is only one type of reduction that can be applied.

Experimental Results

We ran all experiments on a PC workstation with a 2.0 GHZ Xeon CPU and 2 GB memory. We measured total running time required, including that for parsing, preprocessing and SAT solving. We used Precosat (build236) (Biere 2009), the winner of the application track in the SAT'09 competition, as the SAT solver for all planners that we tested and compared. The planners considered include SatPlan06 (Kautz, Selman, and Hoffmann 2006), SatPlan06^L and SASE. SatPlan06 used is the original planner, only with the underlying SAT solver changed. We also implemented long-distance mutual exclusion (londex) (Chen et al. 2009) in SatPlan06; the enhanced solver is denoted as SatPlan06^L. We compared against londex since it also derives transition information from the SAS+ formalism.

We tested all problem instances of STRIPS domains in the 4th and 5th International Planning Competition (IPCs). Some domains, for example, PSR and PROMELA, are not included because all instances are solvable to all three planners within 10 seconds. We used the parser by Fast-Downward (Helmert 2006) to generate SAS+ formalisms from STRIPS inputs. The preprocessing and encoding parts of SASE are implemented in Python2.6.

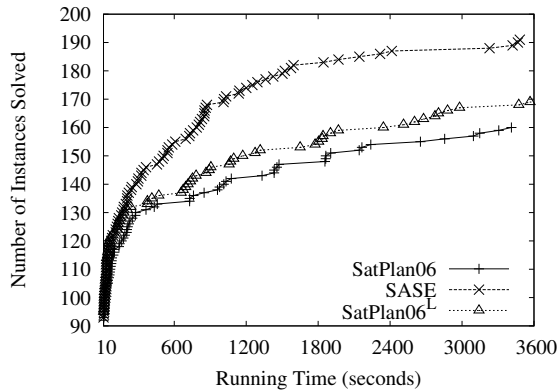


Figure 2: Number of instances (out of all the instances in the testing domains listed in Table 4) that are solvable for a given time limit.

In Figure 2, we present the number of instances that are solvable in the testing domains, as listed in Table 4, with respect to a given time limit. Comparing to SatPlan06, SatPlan06^L solved a moderate number of more instances. SASE in general solved many more instances.

Table 3 presents the number of instances solved in each individual domain. The time limit for each instance was set to 3600 seconds. SASE solves more instances in most domains, especially in Airport and Pipesworld. In Storage, as the only exception, SASE solves one fewer instance than both SatPlan06 and SatPlan06^L.

Table 4 gives more details on some of the instances considered. Due to space limit, since SatPlan06^L has the same number of variables as SatPlan06, we omit in the table the numbers of variables and clauses for SatPlan06^L. We list two largest solvable instances in each domain. If both of the two largest instances are solved by only one planner, we add

Domains	SatPlan06	SatPlan06 ^L	SASE
Openstack	5	5	5
Pathway	12	12	13
Pipesworld	15	16	23
Rovers	13	13	14
Storage	16	16	15
TPP	27	28	29
Trucks	5	5	8
Airport	34	38	46
Driverslog	16	16	16
Freecell	4	5	6
Zeno	15	15	16

Table 3: Number of instances solved in each domain.

one more instance which was solved by at least one more planner. For example, only SASE solved the two largest instances of Pipesworld, Instances 18 and 27. To get information for time and memory comparison, we add one more instance, Pipesworld-12, which was solved by both SatPlan06 and SatPlan06^L. We do not present any instance solved by all planners in less than 100 seconds. For instance, all instances in both Storage and Rovers, except one in each of these domains, were solved by all planners within 100 seconds, thus only the largest instances are presented.

Besides the three planners, in Table 4 we also present the solving time (column Time^b), and the memory consumption (column Mem^b) of SASE, when mutex clique representation is not used. Without mutex clique representation, SASE in general has comparable running times but much larger memory consumptions than the original SASE.

From Table 5, it is evident that SASE is more scalable than the other two planners compared. Furthermore, SASE in general uses fewer clauses and runs in less memory, even though it typically has more variables. One exception is Zenotravel, on which SASE uses less variables than SatPlan06, because all action variables are reduced in SASE. On most instances that we tested, the amount of memory used by SASE is about 1/10 of that by SatPlan06. Note that the memory consumption is reported by Precosat, the SAT solver, before Precosat starts SAT solving. This difference will be larger if the size is measured by CNF file.

Conclusions and Future Work

We developed a new SAS+ based encoding scheme, called SASE, and showed that it can significantly improve the efficacy of planning as satisfiability in both time and memory required. Comparing to SatPlan06, a state-of-the-art SAT based planner which gains its advantage by including extensive mutual exclusions, our experiments showed that SASE outperforms both SatPlan06 and SatPlan06+londex in most domains tested. The comparison to SatPlan06+londex also revealed that the problem structure encoded and the state variables (invariants) used by SASE play key roles in achieving its efficiency. Different from the earlier STRIPS-based encodings, including those using split-action representations, in SASE *actions no longer participate in the constraint propagation over time steps*. We believe this is a major feature makes SASE have a shorter refutation length during SAT solving in practise. Other differences include,

Instances	N	SatPlan06					SatPlan06 ^L			SASE							
		Time	Var	Clause	Mem	t ⁺	Time	Mem	t ⁺	Time	Var	Clause	Mem	t ⁺	Time ^b	Mem ^b	
Openstack-4	23	212.1	3,709	66,744	5	79.3	198.3	5	72.3	33.6	4,889	20,022	2	7.3	23.4	2	
Openstack-5	23	176.3	3,709	66,744	5	64.4	176.7	5	54.7	35.6	5,994	26,367	2	7.6	23.4	2	
Pathway-13	18	3411.8	21,043	660,942	46	3242.1	3567.3	46	3449.5	1573.6	42,157	308,716	21	1362.6	Time Out		
Pathway-17	21	Time Out					Time Out			2316.8	76,483	531,750	37	1392.9	2680.1	92	
Pipesworld-12	16	3147.3	30,078	13,562,157	854	1240.1	2603.5	867	907.3	543.7	43,528	634,873	44	362.0	2126.4	453	
Pipesworld-18	16	Time Out					Time Out			536.3	90,970	1,572,884	104	166.0	676.7	1510	
Pipesworld-27	14	MLE					MLE			1510.8	162,856	3,522,627	231	287.8	MLE		
Rover-18	12	Time Out					Time Out			1384.2	32,304	312,216	22	1195.7	370.5	68	
Storage-16	11	1904.5	13,141	1,737,689	111	1618.4	1316.9	113	1085.8	Time Out						MLE	
TPP-21	12	Time Out					Time Out			1251.7	61,802	427,058	31	1140.1	1340.0	69	
TPP-30	11	3589.7	97,155	7,431,062	462	323.3	1774.1	466	238.9	1844.8	136,106	997,177	70	150.8	2008.3	201	
Trucks-5	19	Time Out					Time Out			326.1	53,521	402,952	29	84.9	380.9	37	
Trucks-7	18	1076.0	21,745	396,581	27	265.2	1069.9	27	231.2	245.7	35,065	255,020	18	65.4	320.6	31	
Trucks-8	19	Time Out					Time Out			830.7	74,500	561,349	39	267.9	715.3	98	
Airport-44	68	Time Out					2977.8	1077	1260.4	852.2	552,670	4,843,879	291	199.6	MLE		
Airport-46	68	Time Out					Time Out			1293.5	716,635	6,698,814	411	302.5	MLE		
Airport-47	68	Time Out					Time Out			1969.7	880,069	8,978,491	512	541.1	MLE		
Driverslog-17	13	2164.8	61,915	2,752,787	183	822.0	1833.7	204	801.8	544.1	74,680	812,312	56	279.7	2748.5	583	
Freecell-5	16	Time Out					2343.8	1197	893.0	1143.1	58,455	989,638	65	534.6	2308.3	403	
Freecell-6	15	Time Out					Time Out			376.1	63,005	1,097,342	80	248.3	1185.5	555	
Zeno-14	6	728.4	26,201	6,632,923	421	83.1	236.8	423	79.8	58.7	17,459	315,719	18	5.4	125.4	95	
Zeno-16	7	Time Out					Time Out			1015.7	40,149	869,827	63	660.6	2214.3	438	

Table 4: Detailed results on various of instances. Column ‘N’ is the optimal time step. Column ‘Time’ is the total running time. Columns ‘Var’, ‘Clause’, ‘Mem’ are the number of variables, number of clauses and memory consumption (in Megabytes), respectively, of the largest SAT encoding. Column ‘t⁺’ is the longest running time among all the SAT formulas. Column Time^b and Mem^b are the running time and memory consumption, when SASE does not use the mutex clique representation. ‘MLE’ is short for memory limit exceeded.

but not limited to, how mutual exclusions and frame axioms are handled. As a result of modeling an hierarchical search space, SASE is able to solve many large benchmark problems that were not solved by the two other planners.

For future research, we plan to study how to utilize other related techniques, for example, planning graph mutual exclusions, stronger additional constraints, and more compact action representation, to improve SASE. It is also a promising future work to further reduce the encoding size. A possible way is to use log-style variables instead of plain ones for transitions. Finally, given the efficiency of SASE, we are looking forward to applying this encoding scheme to other SAT-based planning approaches, such as those for complex planning with preferences or temporal features.

Acknowledgment

The research was supported by NSF grants IIS-0535257, DBI-0743797, IIS-0713109, and a Microsoft Research New Faculty Fellowship.

References

Bäckström, C., and Nebel, B. 1996. Complexity results for sas+ planning. *Computational Intelligence* 11:625–655.

Biere, A. 2009. Pr{e,i}coSAT. In *SAT’09 Competition*.

Blum, A., and Furst, M. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90:1636–1642.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2003. SAT-based planning in complex domains:Concurrency, constraints and nondeterminism. *Artificial Intelligence* 147:85–117.

Chen, Y.; Huang, R.; Xing, Z.; and Zhang, W. 2009. Long-distance mutual exclusion for planning. *Artificial Intelligence* 173:197–412.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic SAT-compilation of planning problems. In *Proc. of IJCAI*.

Helmert, M. 2006. The Fast Downward planning system. *J. of AI Research* 26:191–246.

Hoffmann, J.; Kautz, H.; Gomes, C.; and Selman, B. 2007. SAT encodings of state-space reachability problems in numeric domains. In *Proc. of IJCAI*.

Huang, R.; Chen, Y.; and Zhang, W. 2009. An Optimal Temporally Expressive Planner: Initial Results and Application to P2P Network Optimization. In *Proc. of ICAPS*.

Kautz, H.; Selman, B.; and Hoffmann, J. 1999. Unifying sat-based and graph-based planning. In *Proc. of IJCAI*.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as Satisfiability. In *Abstracts IPC5*.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proc. of 23rd AAAI*.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 12-13:1031–1080.

Rintanen, J. 2006. Biclique-based representations of binary constraints for making SAT planning applicable to larger problems. In *Proc. of ECAI*.

Robinson, N.; Gretton, C.; Pham, D.; and Sattar, A. 2009. SAT-Based Parallel Planning Using a Split Representation of Actions. In *Proc. of 19th ICAPS*.