Exploiting optimizations in SAT-based planning: minimal-#actions plans and “soft” goals

Enrico Giunchiglia and **Marco Maratea**

Laboratory of Systems and Technologies for Automated Reasoning (STAR-Lab)
DIST - University of Genova

Roma, July 5-6 2007

Context

SAT-based planning is the best approach for “optimally” solve planning problems by

1. constructing a SAT formula ϕ_n for a fixed *makespan* n;
2. verifying if ϕ_n is satisfiable; if not, n is increased.

Main advantages:

- simplicity
- effectiveness, can take advantage on the continuous progress in the SAT area
- optimal makespan guaranteed

International Planning Competitions

SATPLAN has been the winner of the IPC-4 and co-winner of the IPC-5 in the optimal tracks.

On the other hand . . .

SATPLAN's deficiency

- it can only handle a very limited part of the PDDL language; and
- it does not take into account other "plan quality" issues, e.g., number of actions in the plan and the possibility to express "soft" goals.

On the other hand . . .

SATPLAN's deficiency

- it can only handle a very limited part of the PDDL language; and
- $\surd$ it does not take into account other "plan quality" issues, e.g., number of actions in the plan and the possibility to express "soft" goals.

Contribution of the work

We present SATPLANP, modification of SATPLAN, which returns plans

1. having minimal number of actions;
2. having maximal number of “soft” goals satisfied.

w.r.t. both

- subset inclusions (qualitative)
- cardinality (quantitative)

Planning as Satisfiability

Planning problem

Is a triple $\langle I, tr, G\rangle$ where (given the sets of fluents $\mathcal{F}$ and actions $\mathcal{A}$)

- I is a SAT formula over $\mathcal{F}$ and represents the set of *initial states*;
- tr is a SAT formula over $\mathcal{F} \cup \mathcal{A} \cup \mathcal{F}'$ where $\mathcal{F}' = \{f' : f \in \mathcal{F}\}$ is a copy of the fluent signature and represents the *transition relation*
- G is a SAT formula over $\mathcal{F}$ and represents the set of *goal states*.

Plan

The *planning problem* Π *with makespan* n is the SAT formula Π_n

$$I_0 \wedge \wedge_{i=1}^{n} tr_i \wedge G_n \qquad (n \geq 0) \tag{1}$$

- tr_i is the formula obtained from tr by substituting each symbol $p \in \mathcal{F} \cup \mathcal{A}$ with p_{i-1} and each $f \in \mathcal{F}'$ with f_i
- I_0 and G_n are obvious

A *plan for* Π_n is an interpretation satisfying (1).

```
function SATPLAN(Π_n)
    return DLL(cnf(Π_n),∅)

function DLL(φ,S)
    if (∅ ∈ φ) return FALSE;
    if (φ = ∅) return S;
    if ({l} ∈ φ) return DLL(φ_l, S ∪ {l});
    l := ChooseLiteral(φ);
    return DLL(φ_l̄, S ∪ {l̄}) or
    return DLL(φ_l̄, S ∪ {l}).
```

φ_l

φ_l returns the formula obtained from φ by (*i*) deleting the clauses containing l, and (*ii*) deleting $\bar{l}$ from the others.

Qualitative SATPLANP's algorithm

function QL-SATPLANP(Π_n,P,$\prec$)
  return OPT-DLL($cnf(\Pi_n \wedge \wedge_{p \in P}(v(p) \equiv p))$,$\emptyset$,$v(P)$,$v(\prec)$)

function OPT-DLL(φ,S,P',$\prec'$)
  if $(\emptyset \in \varphi)$ **return** FALSE;
  if $(\varphi = \emptyset)$ **return** S;
  if $(\{l\} \in \varphi)$ **return** OPT-DLL($\varphi_l, S \cup \{l\}, P', \prec'$);
  $l := ChooseLiteral(\varphi, S, P', \prec')$;
  $V :=$ OPT-DLL($\varphi_l, S \cup \{l\}, P', \prec'$);
  if ($V \neq$ FALSE) **return** V;
  return OPT-DLL($\varphi_{\bar{l}}, S \cup \{\bar{l}\}, P', \prec'$).

where

- for each $p \in P$, $v(p)$ is a newly introduced variable;
- $v(P)$ is the set of new variables, i.e., $\{v(p) : p \in P\}$;
- $v(\prec) = \prec'$ is the partial order on $v(P)$ defined by $v(p) \prec' v(p')$ iff $p \prec p'$;

Qualitative SATPLANP's algorithm: Example

Going at work from home

$$\neg AtWork_0, \quad AtWork_1 \equiv \neg AtWork_0 \equiv (Car_0 \vee Bus_0 \vee Bike_0), \quad AtWork_1, \tag{2}$$

If we have two preferences

1. $p_1 = (\neg Bike_0 \wedge \neg Bus_0 \wedge \neg Car_0)$
2. $p_2 = (\neg Bike_0 \wedge \neg Bus_0)$

with $p_1 \prec p_2$. OPT-DLL on (2) returns the plan corresponding to $\{Car_0\}$ determined while exploring the branch extending $\{\neg v(p_1), v(p_2)\}$.

function QT-SATPLANP(Π_n,P,c)
16 **return** OPT-DLL($cnf(\Pi_n \wedge adder(P,c))$,$\emptyset$,$b(c)$,$p(c)$)

$adder(P,c)$ is a SAT formula, e.g., (Warners, IPL 1999)

- if $n = \lceil log_2((\sum_{p \in P} c(p)) + 1) \rceil$, $adder(P,c)$ contains n new variables $\{b_{n-1}, \ldots, b_0\} = b(c)$; and
- for any plan π satisfying Π_n, there exists a unique interpretation μ to the variables in $\Pi_n \wedge adder(P,c)$ s.t.
  1. μ extends π and satisfies $\Pi_n \wedge adder(P,c)$;
  2. $\sum_{p \in P: \pi \models p} c(p) = \sum_{i=0}^{n-1} \mu(b_i) \times 2^i$, where $\mu(b_i)$ is 1 if μ assigns b_i to true, and is 0 otherwise.
- $p(c)$ is the partial order $b_{n-1} \prec b_{n-2} \prec \cdots \prec b_0$.

Quantitative SATPLANP's algorithm: Example

Going at work from home

$$\begin{array}{c} \neg AtWork_0, \\ AtWork_1 \equiv \neg AtWork_0 \equiv (Car_0 \vee Bus_0 \vee Bike_0), \\ AtWork_1, \end{array} \quad (3)$$

If we have two preferences

1. $p_1 = (\neg Bike_0 \wedge \neg Bus_0 \wedge \neg Car_0)$
2. $p_2 = (\neg Bike_0 \wedge \neg Bus_0)$

with $c(p_1) = 2$ and $c(p_2) = 1$, then two bits b_1 and b_0 are sufficient as output of $adder(\{p_1, p_2\}, c)$. OPT-DLL returns the plan corresponding to $\{Car_0\}$ determined while exploring the branch extending $\neg b_1, b_0$.

Rational and utility of SATPLANP's algorithms

Rational

- we preferentially (and in order) split on variables (literals) defining
  1. preferences, in the qualitative case
  2. the sum of the weights of preferences, in the quantitative case
- for minimizing actions, the preferences are the atoms related to actions and the split is forced to FALSE
- for maximizing soft goals satisfied, the preferences are the soft goals and the split is forced to TRUE

Utility

- if we want that as few as possible actions are executed, then we have to find an assignment in QT-SATPLANP;
- if we want that no redundant sequence of (possibly parallel) actions is executed, then we have to find an assignment in QL-SATPLANP.

Rational and utility of SATPLANP's algorithms

Rational

- we preferentially (and in order) split on variables (literals) defining
  1. preferences, in the qualitative case
  2. the sum of the weights of preferences, in the quantitative case
- for minimizing actions, the preferences are the atoms related to actions and the split is forced to FALSE
- for maximizing soft goals satisfied, the preferences are the soft goals and the split is forced to TRUE

Utility

- if we want that as few as possible actions are executed, then we have to find an assignment in QT-SATPLANP;
- if we want that no redundant sequence of (possibly parallel) actions is executed, then we have to find an assignment in QL-SATPLANP.

Rational and utility of SATPLANP's algorithms

Rational

- we preferentially (and in order) split on variables (literals) defining
  1. preferences, in the qualitative case
  2. the sum of the weights of preferences, in the quantitative case
- for minimizing actions, the preferences are the atoms related to actions and the split is forced to FALSE
- for maximizing soft goals satisfied, the preferences are the soft goals and the split is forced to TRUE

Utility

- if we want that as few as possible actions are executed, then we have to find an assignment in QT-SATPLANP;
- if we want that no redundant sequence of (possibly parallel) actions is executed, then we have to find an assignment in QL-SATPLANP.

Experimental analysis: Goals

1. Evaluate SATPLANP w.r.t. the state-of-the-art on problems with "SimplePreferences", like "soft" goals
   - SATPLANP vs. SGPLAN
2. Evaluate the reductions that can be obtained with SATPLANP over SATPLAN
3. Evaluate the computational costs of such reductions
   - SATPLANP vs. SATPLAN
4. Evaluate what kind of *adder*() works better
   - (Warners, IPL 1999) vs. (Bailleaux & Boufkhad, CP 2003)

Experimental analysis: Goals

1. Evaluate SATPLANP w.r.t. the state-of-the-art on problems with "SimplePreferences", like "soft" goals
   - SATPLANP vs. SGPLAN
2. Evaluate the reductions that can be obtained with SATPLANP over SATPLAN
3. Evaluate the computational costs of such reductions
   - SATPLANP vs. SATPLAN
4. Evaluate what kind of *adder*() works better
   - (Warners, IPL 1999) vs. (Bailleaux & Boufkhad, CP 2003)

Experimental analysis: Goals

1. Evaluate SATPLANP w.r.t. the state-of-the-art on problems with "SimplePreferences", like "soft" goals
   - SATPLANP vs. SGPLAN
2. Evaluate the reductions that can be obtained with SATPLANP over SATPLAN
3. Evaluate the computational costs of such reductions
   - SATPLANP vs. SATPLAN
4. Evaluate what kind of *adder*() works better
   - (Warners, IPL 1999) vs. (Bailleaux & Boufkhad, CP 2003)

Experimental analysis: Goals

1. Evaluate SATPLANP w.r.t. the state-of-the-art on problems with "SimplePreferences", like "soft" goals
   - SATPLANP vs. SGPLAN
2. Evaluate the reductions that can be obtained with SATPLANP over SATPLAN
3. Evaluate the computational costs of such reductions
   - SATPLANP vs. SATPLAN
4. Evaluate what kind of *adder*() works better
   - (Warners, IPL 1999) vs. (Bailleaux & Boufkhad, CP 2003)

Experimental analysis: Goals

1. Evaluate SATPLANP w.r.t. the state-of-the-art on problems with “SimplePreferences”, like “soft” goals
   - SATPLANP vs. SGPLAN
2. Evaluate the reductions that can be obtained with SATPLANP over SATPLAN
3. Evaluate the computational costs of such reductions
   - SATPLANP vs. SATPLAN
4. Evaluate what kind of *adder*() works better
   - (Warners, IPL 1999) vs. (Bailleaux & Boufkhad, CP 2003)

Experimental analysis: Goals

1. Evaluate SATPLANP w.r.t. the state-of-the-art on problems with "SimplePreferences", like "soft" goals
   - SATPLANP vs. SGPLAN
2. Evaluate the reductions that can be obtained with SATPLANP over SATPLAN
3. Evaluate the computational costs of such reductions
   - SATPLANP vs. SATPLAN
4. Evaluate what kind of *adder*() works better
   - (Warners, IPL 1999) vs. (Bailleaux & Boufkhad, CP 2003)

Experimental analysis: 1.

	SGPLAN	SATPLAN	SATPLANP(w)	SATPLANP(b)	SATPLANP(s)
pipe	0/0	0/7	0/18	0/18	0/17
pipet	0/0	0/5	0/11	0/11	0/11
sat	0/10	0/4	0/4	0/4	0/4
air	0/23	0/9	0/11	0/11	0/11
phil	29/0	0/29	0/464	0/464	0/464
opt	12/0	0/12	0/90	0/90	0/90
psr	12/157	0/48	0/231	0/231	0/231
dep	2/3	0/4	0/7	0/7	0/7
driv	0/71	0/10	0/54	0/54	0/50
zeno	0/44	0/9	0/24	0/24	0/24
free	0/12	0/3	0/8	0/8	0/8
log	0/51	0/10	0/33	0/33	0/33
block	0/33	0/9	0/12	0/12	0/12
mpr	0/0	0/4	0/4	0/4	1/3
myst	0/0	0/2	0/2	0/2	0/2
path	7/0	0/7	0/21	0/21	0/21
stor	0/30	0/9	0/10	0/10	0/10
TPP	5/14	0/9	0/27	0/27	0/27
truck	3/0	0/3	0/3	0/3	0/3
Total	70/448	0/193	0/1034	0/1034	1/1028

Table: Results on domains coming from IPCs. x/y stands for x time outs or segmentation faults, y soft goals satisfied.

Experimental analysis: 2. and 4.

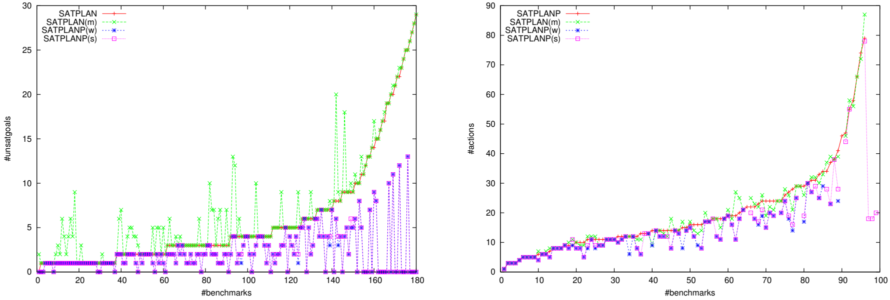
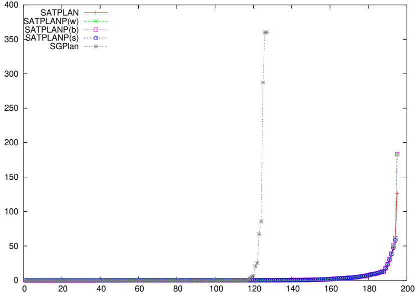


Figure: Left: Number of unsatisfied soft goals by SATPLAN, SATPLAN(m), and SATPLANP(w)/(s). Right: Number of actions in the returned plan for SATPLAN, SATPLAN(m), and SATPLANP(w)/(s).

Experimental analysis: 3. and 4.



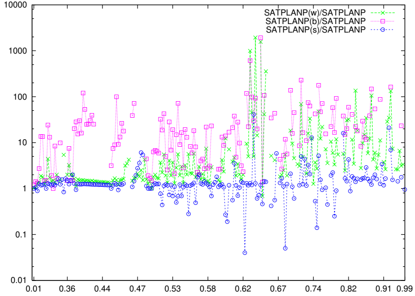


Figure: Left: Performances of SGPLAN, SATPLANP/(w)/(b)/(s). Right: Performances of SATPLANP(w)/(b)/(s) w.r.t. SATPLAN as a function of the ration between the number of preferences and the number of variables.

Further experiments

	#actions					Makespan	
PB	SATPLAN	SATPLAN(m)	SATPLANP(w)	SATPLANP(s)	SGPLAN	SATPLANP	SGPLAN
log6-0	33	30	25	25	26	9	26
log6-1	28	21	14	16	15	9	15
log6-9	41	39	24	28	28	11	28
block6-0	12	12	12	12	12	12	12
block6-1	10	10	10	10	16	10	16
block6-2	20	20	–	20	32	20	32
stor9	14	14	12	12	11	7	11
stor11	–	–	–	18	17	11	17
stor12	22	25	–	20	17	9	17
sat1	9	9	9	9	9	8	9
sat2	13	–	–	13	13	12	13
psr33	21	21	21	21	21	16	21
psr40	20	20	20	–	20	15	20
psr42	30	30	30	30	30	16	30
driv1	14	18	8	8	7	6	7
zeno5	15	14	14	14	11	5	11
zeno6	14	13	12	12	13	5	13
zeno8	16	17	15	15	12	5	12
free1	9	11	9	11	10	5	10
free2	18	18	–	18	14	8	14
free3	21	21	21	21	19	7	19
air7	41	41	41	41	41	21	41
air9	71	71	–	71	73	27	73
air12	39	39	39	39	39	21	39

Table: #actions and makespan for SATPLAN, SATPLANP and SGPLAN.

Conclusions and future work

Done . . . We have

- √ extended SAT-based planning to deal with (other) issues related to plan quality;
- √ showed that our approach is viable and competitive, often without sacrificing efficiency;
- √ evaluated different minimalities and *adder*()s

To be done . . .

- × relaxe the optimality in the makespan for further improve the plan quality;
- × experiment with other encodings, other than the "action-based".

More on this work . . .

- SATPLANP's web page at
  `http://www.star.dist.unige.it/~marco/SATPLANP/`
- Our ECAI 2006 and AAAI 2007 papers:
  - "Solving Optimization Problems with DLL"; and
  - "Planning as Satisfiability with Preferences"
- Enrico's invited talk at ICAPS'06