

Satisfiability and Preferences: Solving Optimization Problems with DLL

Marco Maratea

j.w.w. Enrico Giunchiglia

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

L'Aquila, 10 Apr 2008

SAT: The problem

A *literal* l is a proposition p or its negation $\neg p$ ($\neg\neg p \equiv p$).

Given the literals l_1, \dots, l_k , a *clause* is $l_1 \vee \dots \vee l_k$.

Given the clauses c_1, \dots, c_m , a CNF (Conjunctive Normal Form) formula is $c_1 \wedge \dots \wedge c_m$.

An *assignment*, or *valuation* v , is a partial function from the propositions to $\{\text{TRUE}, \text{FALSE}\}$.

We can extend the definition of v in the natural way to assign truth values to literals, clauses and formulas.

Given a CNF formula φ , we define the *propositional satisfiability problem (SAT)*:

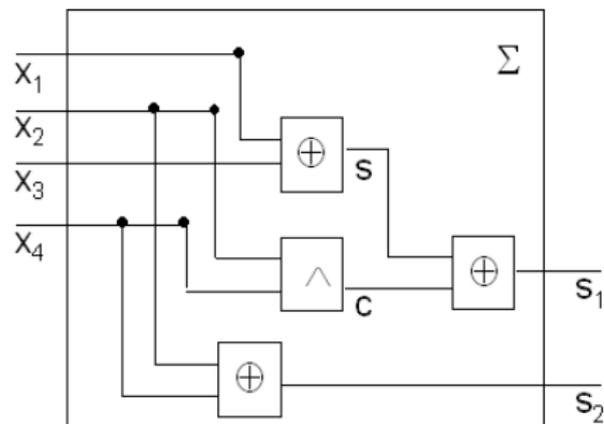
Does there exist an assignment v to the propositions in φ such that φ is true?

- 1 $\varphi := \{p, p \vee \neg q, \neg r\}$ has the (total) satisfying assignments
 - $\{p := \text{TRUE}, q := \text{TRUE}, r := \text{FALSE}\}$
 - $\{p := \text{TRUE}, q := \text{FALSE}, r := \text{FALSE}\}$
- 2 $\varphi := \{\neg p, p \vee \neg q, r \vee \neg p, q\}$ has no satisfying assignments because the clause $\{p \vee \neg q\}$ can not be satisfied.

The complexity of SAT

- Prototypical NP-complete problem
- n variables imply 2^n possible interpretations (total assignments) from which a satisfying one has to be guessed
- A SAT solver is a program that tries to find satisfying interpretations automatically

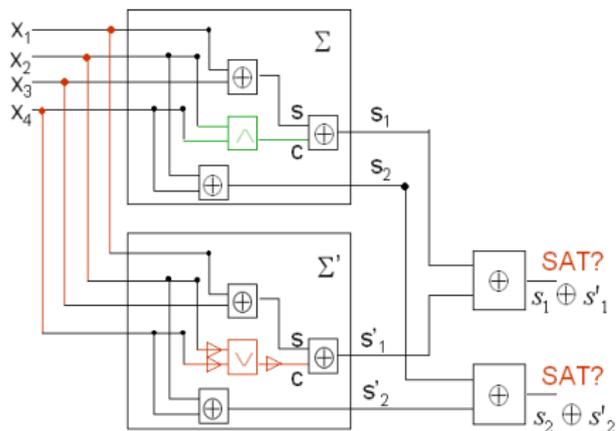
Formulas and circuits



$$s_1 = \overbrace{((X_1 \oplus X_3))}^s \oplus \overbrace{(X_2 \wedge X_4)}^c$$
$$s_2 = (X_2 \oplus X_4)$$

input signal	variable
gate	operator
inverter	↔ not
signal	subformula
output	formula value

Example I: Equivalence verification



Setting

- Σ is the abstract model
- Σ' is the implementation

Miter-based FCEV

SAT of $(s_1 \oplus s'_1) \vee (s_2 \oplus s'_2)$

Example II: AI Planning

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 (Kautz and Selman, ECAI 1992)

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

$$I_0 \wedge \bigwedge_{i=1}^n tr_i \wedge G_n \quad (n \geq 0) \quad (1)$$

- 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).

- Resolution algorithm
- Local search algorithms
- (Ordered) Binary Decision Diagrams (OBDDs)
- Stalmark's method
- Davis-Logemann-Loveland (DLL) algorithm

The DLL algorithm

```
function SAT( $\phi$ ) return DLL(CNF( $\phi$ ),  $\emptyset$ );
```

```
function DLL( $\varphi$ ,  $S$ )
```

```
  if  $\varphi = \emptyset$  then return TRUE ;
```

```
  if  $\emptyset \in \varphi$  then return FALSE ;
```

```
  if  $\{l\} \in \varphi$  then return DLL( $\varphi_l$ ,  $S \cup \{l\}$ );
```

```
   $A :=$  an atom occurring in  $\varphi$ ;
```

```
  return DLL( $\varphi_A$ ,  $S \cup \{A\}$ ) or
```

```
    DLL( $\varphi_{\neg A}$ ,  $S \cup \{\neg A\}$ ).
```

φ_l

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

Theorem (Davis, Logemann and Loveland, JACM 1962)

SAT(ϕ) returns TRUE if ϕ is satisfiable, and FALSE otherwise.

- DLL is a decision procedure: Given a formula φ , it can decide whether it is satisfiable, or not.
- In many cases such an information is not enough.
- The solution has also to be “optimal” in some sense, e.g., it has to minimize/maximize a given objective function
- Example: (classical) Planning with action costs

Goal of the talk

- 1 Present a simple model for qualitative preferences on Boolean variables
- 2 Present two algorithms for solving satisfiability problems with preferences
- 3 Show how optimal models can be computed via simple modifications to DLL
- 4 Show how MIN-ONE, MAX-SAT and other problems can be recasted and solved in the framework
- 5 Show the viability of the approaches with experimental analysis

Assignments and optimal models

Assignment

- An **assignment** μ is a consistent set of literals
- An assignment μ is **total** if it is maximally consistent
- An assignment μ **satisfies** (or is a **model** of) a formula φ if for each $C \in \varphi$, $\mu \cap C \neq \emptyset$

Setting

- 1 \prec is a partial order on the set of total assignments satisfying the formula φ .
- 2 Intuitively, $\mu \prec \mu'$ means that μ is **preferred** to μ' .

Optimality of an assignment

- 1 A model μ is **optimal** (for φ) if there exists a total assignment extending μ which is a minimal element of the partial order.

From \prec on literals to \prec on models

Setting

A *qualitative preference* is a pair $\langle S, \prec \rangle$ where

- S is a set of literals representing **preferences**; and
- \prec is a partial order on (the literals in) S

$\mu \prec \mu'$

μ and μ' are total assignments. $\mu \prec \mu'$ if and only if

- 1 there exists a literal $l \in S$ with $l \in \mu$ and $\bar{l} \in \mu'$; and
- 2 $\forall l' \in S \cap (\mu' \setminus \mu), \exists l \in S \cap (\mu \setminus \mu')$ such that $l \prec l'$.

Example

$S = \{\bar{x}_0, \bar{x}_1\}$ with $\bar{x}_1 \prec \bar{x}_0$. Then,

- 1 $\{\bar{x}_1, \bar{x}_0\} \prec \{\bar{x}_1, x_0\} \prec \{x_1, \bar{x}_0\} \prec \{x_1, x_0\}$;
- 2 if φ is $(x_0 \vee x_1)$, the optimal model is $\{\bar{x}_1, x_0\}$.

```

function OPT-DLL( $\varphi, \mu, \mathcal{S}, \prec$ )
  1 if ( $\emptyset \in \varphi$ ) return FALSE;
  2 if ( $\varphi = \emptyset$ ) return  $\mu$ ;
  3 if ( $\{I\} \in \varphi$ ) return OPT-DLL( $\varphi_I, \mu \cup \{I\}, \mathcal{S}, \prec$ );
  4  $I :=$  a minimal unassigned variable in  $\mathcal{S}, \prec$  if  $\exists$ 
     and a literal in  $\varphi$  otherwise.
  5  $v :=$  OPT-DLL( $\varphi_I, \mu \cup \{I\}, \mathcal{S}, \prec$ );
  6 if ( $v \neq \text{FALSE}$ ) return  $v$ ;
  7 return OPT-DLL( $\varphi_{\bar{I}}, \mu \cup \{\bar{I}\}, \mathcal{S}, \prec$ ).

```

Theorem (Giunchiglia and Maratea, ECAI 2006)

OPT-DLL($\varphi, \emptyset, \mathcal{S}, \prec$) returns an optimal model of φ wrt $\langle \mathcal{S}, \prec \rangle$ if φ is satisfiable, and FALSE otherwise.

Qualitative preferences: Example

Preference

- 1 $\{\neg Bus_0, \neg Bike_0\}, \{\neg Bus_0 \prec \neg Bike_0\}$

OPT-DLL looks for a model extending

- 1 $\{\neg Bus_0, \neg Bike_0\}$; if no such model exists, OPT-DLL looks for a model extending
- 2 $\{\neg Bus_0, Bike_0\}$; if no such model exists, OPT-DLL looks for a model extending
- 3 $\{Bus_0, \neg Bike_0\}$; if no such model exists, OPT-DLL looks for a model extending
- 4 $\{Bus_0, Bike_0\}$; if no such model exists, OPT-DLL returns false.

Formulas

- 1 $AtWork_1 \equiv \neg AtWork_0 \equiv (Car_0 \vee Bus_0 \vee Bike_0)$
- 2 $\neg AtWork_0 \wedge AtWork_1$

$\{Car_0, AtWork_1\}$ is the optimal model.

Quantitative preferences

Idea

Quantitative preferences are reduced to qualitative.

A quantitative preference is a pair $\langle S, c \rangle$ where

- S is the set of preferences;
- c is a function which maps each element of S to a positive integer number (a *weight*).

Adder and solving algorithm

- 1 an assignment μ is optimal if it maximizes the *utility function* defined as $\sum_{I \in \mu \cap S} c(I)$
- 2 $adder(S, c)$ is an *adder* for S with output b_{n-1}, \dots, b_0 , ($n = \lceil \log_2(\sum_{I \in S} c(I) + 1) \rceil$) s.t. $\sum_{I \in S \cap \mu} c(I) = \sum_{i=0}^{n-1} \mu'(b_i) \times 2^i$, where $\mu'(b_i)$ is 1 if $b_i \in \mu'$, and is 0 otherwise. It is implemented in linear time using the encoding of (Warners, ILP 1998).
- 3 $OPT-DLL(\varphi \cup adder(S), \emptyset, \{b_{n-1}, \dots, b_0\}, b_{n-1} \prec \dots \prec b_0)$ returns an optimal model for φ wrt $\langle S, 1 \rangle$ if φ is satisfiable, and FALSE otherwise.

Quantitative preferences: Example

Formulas

- 1 $AtWork_1 \equiv \neg AtWork_0 \equiv (Car_0 \vee Bus_0 \vee Bike_0)$
- 2 $\neg AtWork_0 \wedge AtWork_1$

If we have the preferences:

- 1 $\{\neg Bike_0, \neg Bus_0, \neg Car_0\}$, assuming the reward function c is constant and different from 0, then the optimal models are $\{Bike_0, AtWork_1\}$, $\{Bus_0, AtWork_1\}$, $\{Car_0, AtWork_1\}$.
- 2 $\{\neg Bike_0, \neg Bus_0, \neg Car_0\}$, if we assume $c(\neg Bike_0) = 2$ while $c(\neg Bus_0) = c(\neg Car_0) = 1$, then the optimal models are $\{Bus_0, AtWork_1\}$ and $\{Car_0, AtWork_1\}$.

Setting

φ is a formula; P is the set of variables in φ ; μ, μ' total assignments.

MIN-ONE and $\text{MIN-ONE}_{\subseteq}$

- 1 In MIN-ONE problems, $\mu \prec \mu'$ if $|\mu \cap P| < |\mu' \cap P|$.
- 2 In $\text{MIN-ONE}_{\subseteq}$ problems, $\mu \prec \mu'$ if $\mu \cap P \subset \mu' \cap P$.

Solving MIN-ONE and MIN-ONE_⊆ with OPT-DLL

Setting

φ is a formula; P is the set of variables in φ ; $\bar{P} = \{\bar{x} : x \in P\}$

MIN-ONE_⊆

OPT-DLL($\varphi, \emptyset, \bar{P}, \emptyset$) returns an optimal assignment if φ is satisfiable, and FALSE otherwise.

MIN-ONE

OPT-DLL($\varphi \cup \text{adder}(P), \emptyset, \{b_{n-1}, \dots, b_0\}, \bar{b}_{n-1} \prec \dots \prec \bar{b}_0$) returns an optimal model if φ is satisfiable, and FALSE otherwise.

Setting

φ is a formula; μ, μ' are total assignments.

MAX-SAT and MAX-SAT \subseteq

- 1 In MAX-SAT \subseteq problems, $\mu \prec \mu'$ if the set of clauses satisfied by μ is a superset of the clauses satisfied by μ'
- 2 In MAX-SAT problems, $\mu \prec \mu'$ if the cardinality of the set of clauses satisfied by μ is bigger than the cardinality of the set of the clauses satisfied by μ'

From MAX-SAT and MAX-SAT_⊆ to MIN-ONE and MIN-ONE_⊆

Intuition

- 1 Given a formula φ , a new variable v_i is added to the clause $C_i \in \varphi$, e.g., $\{\{a\}, \{b, c\}\}$ becomes $\{\{a, v_1\}\{b, c, v_2\}\}$.
- 2 MAX-SAT and MAX-SAT_⊆ correspond to minimize the set of variables v_i assigned to 1, and thus reduce (with some care) to MIN-ONE and MIN-ONE_⊆ problems.

DISTANCE-SAT and DISTANCE-SAT_⊆

DISTANCE-SAT: Definition (Bailleaux and Marquis, JAR 2006)

In DISTANCE-SAT, given a formula φ and an assignment μ , the goal is to find an assignment μ' satisfying φ and differing “as little as possible” from μ .

DISTANCE-SAT and DISTANCE-SAT_⊆

Let φ be a formula. Let μ be an assignment. Two facts hold:

- 1 Let \prec be a partial order such that $l \prec \bar{l}$ if $l \in \mu$. If $\text{OPT-DLL}(\varphi, \emptyset, \prec)$ returns an assignment μ' then $\mu' \in \text{DISTANCE-SAT}_{\subseteq}(\varphi, \mu)$. If $\text{OPT-DLL}(\varphi, \emptyset, \prec)$ returns FALSE then φ is unsatisfiable.
- 2 Let $\text{adder}(\mu)$ be an adder of μ with output b_{n-1}, \dots, b_0 . Let \prec be a partial order on $\{b_{n-1}, \dots, b_0\}$ such that for each $i \in [0, n-1]$, $b_i \prec \bar{b}_i$, and, if $i \neq 0$, $b_i \prec b_{i-1}$. If $\text{OPT-DLL}(\varphi \cup \text{adder}(\mu), \emptyset, \prec)$ returns an assignment μ' then $\mu' \cap P \in \text{DISTANCE-SAT}(\varphi, \mu)$. If $\text{OPT-DLL}(\varphi \cup \text{adder}(\mu), \emptyset, \prec)$ returns FALSE then φ is unsatisfiable.

By using OPT-DLL, we are guaranteed that the first returned solution is optimal. On the other hand

- We have to impose an ordering on the literals to be followed while branching; such ordering
- implies changes in the DLL algorithm (and related implementation); and
- in the literature, it is well-known that such imposition can lead to significant degradation in performances

Idea

- does not require any modification of DLL heuristic and thus does not have the OPT-DLL (theoretical) disadvantages;
- once a solution is computed, a constraint (i.e., a formula) is added to the input formula imposing that the new solution (if any) will be better than the last computed wrt the qualitative preference on literals expressed.

Preference formulas

Example

$$\varphi = (\neg \text{Fish} \vee \neg \text{Meat}) \wedge (\neg \text{RedWine} \vee \neg \text{WhiteWine})$$

$$\langle S, \prec \rangle = \{\text{Fish}, \text{RedWine}, \text{WhiteWine}\}, \{\text{WhiteWine} \prec \text{RedWine}\}$$

The optimal model is $\{\text{Fish}, \text{WhiteWine}, \neg \text{RedWine}, \neg \text{Meat}\}$.

Preference formula

The *preference formula* for μ wrt S, \prec is

$$(\forall I: I \in S, I \notin \mu) \wedge (\wedge I': I' \in S, I' \in \mu (\forall I: I \in S, I \notin \mu, I \prec I' \vee I')) \quad (2)$$

μ' which satisfies (2) are such that $\mu' \prec \mu$.

Example

Considering the preference above

- if $S_1 = \{\text{Meat}, \text{RedWine}, \neg \text{Fish}, \neg \text{WhiteWine}\}$, then (2) is

$$\psi_1 : (\text{Fish} \vee \text{WhiteWine}) \wedge (\text{WhiteWine} \vee \text{RedWine})$$

The SATPREF decision procedure

$S, \prec :=$ a qualitative preference on literals;

$\varphi :=$ the input formula; $\psi := \top$; $\mu_{opt} := \emptyset$;

function PREF-DLL($\varphi \cup \psi, \mu$)

if $\perp \in (\varphi \cup \psi)_\mu$ **then return** FALSE ;

if μ is total $\mu_{opt} := \mu$; $\psi := Reason(\mu, S, \prec)$; **return** FALSE ;

if $\{I\} \in (\varphi \cup \psi)_\mu$ **then return** PREF-DLL($\varphi \cup \psi, \mu \cup \{I\}$);

return PREF-DLL($\varphi \cup \psi, \mu \cup \{A\}$) **or**

PREF-DLL($\varphi \cup \psi, \mu \cup \{\neg A\}$).

Reason(μ, S, \prec) returns the set of clauses corresponding to the preference formula for S wrt S, \prec ;

Theorem (Di Rosa, Giunchiglia and Maratea, 2008)

PREF-DLL terminates, and then μ_{opt} is empty if φ is unsatisfiable, and an optimal model of φ wrt S, \prec otherwise.



Experimental analysis

	domain	#	OPTSAT	SATPREF	OPBDP	PBS4	MSAT+	BSOLO	MMSAT	OPTSAT	SATPREF
1	MINONE	26	0.69(26)	0.2(26)	85.37(7)	17.56(19)	7.33(24)	115.73(22)	87.21(24)	93.24(24)	23.99(25)
2	Partial MINONE	21	77.99(19)	2.7(21)	—	223.14(15)	43.32(18)	433.21(16)	391.21(12)	74.28(21)	69.89(21)
3	MAXSAT	35	26.68(34)	11.25(35)	20.89(3)	98.55(10)	130.37(31)	192.56(23)	229.73(21)	218.86(31)	175.12(31)
4	MAXCUT-1	5	0.01(5)	0.01(5)	0.99(1)	66.67(1)	0.86(1)	76.57(1)	1.09(3)	7.56(1)	7.52(1)
5	MAXCUT-2	62	0.01(62)	0.01(62)	230.33(5)	0.01(2)	247.54(7)	0.01(2)	194.52(52)	66.86(4)	21.61(3)
6	PSEUDO-1	7	0.02(7)	0.01(7)	2.2(4)	147.58(4)	0.25(5)	30.18(4)	4.75(5)	22.8(5)	36.66(5)
7	PSEUDO-2	17	0.03(17)	0.01(17)	—	85.88(1)	490.36(5)	—	81.93(2)	90.36(3)	338.26(3)
8	PSEUDO-3	148	4.81(130)	0.19(131)	16.65(85)	18.08(90)	11.52 (104)	22.23 (94)	62.08 (107)	31.8(103)	60.59(109)
9	PSEUDO-4	15	11.69(15)	3.12(15)	81.83(5)	102.75(9)	43.74(15)	373.73(8)	109.49(14)	41.49(15)	36.1(15)
10	MAXONE	60	0.96(60)	0.13(60)	296.26(35)	11.48(60)	2.02(58)	40.96(60)	22.5(60)	293(56)	7.87(58)
11	MAXCLIQUE	62	0.01(62)	0.06(62)	70.37(16)	23.79(13)	154.39(22)	248.26(14)	61.97(36)	54.14(19)	178.04(23)

Table: Results for solving satisfiability problems with qualitative (columns 4-5) and quantitative (columns 6-14) preferences. Problems are: MIN-ONE (row 1), partial MIN-ONE (row 2), MAX-SAT (rows 3-5), and partial MAX-SAT (rows 6-11).

A closer look to the performances of SATPREF

	domain	T_1	Q_1	#Calls	T_f	Q_f
1	MINONE	0.19	751.6	2	0.2	751.6
2	Partial MINONE	2.68	45.5	2.5	2.7	44.1
3	MAXSAT	0.05	8605.2	21.2	11.25	8847.6
4	MAXCUT/spinglass	0.01	770.4	2	0.01	770.4
5	MAXCUT/dimacs_mod	0.01	695.9	2.2	0.01	701.9
6	PSEUDO/garden	0.01	496	2	0.01	496
7	PSEUDO/logic-synthesis	0.01	152.2	2	0.01	152.2
8	PSEUDO/primes	0.18	368.4	2	0.19	368.4
9	PSEUDO/routing	3.12	58.7	2	3.12	58.7
10	MAXONE/structured	0.12	240.5	8.4	0.13	249.8
11	MAXCLIQUE/structured	0.06	430.4	2	0.06	430.4

Table: CPU time for finding first (T_1) and optimal (T_f) solution. Numbers of recursive calls to PREF-DLL, including the first ($\#Calls$). Quality of the first (Q_1) and optimal (Q_f) solution.

OPTSAT vs. SATPREF

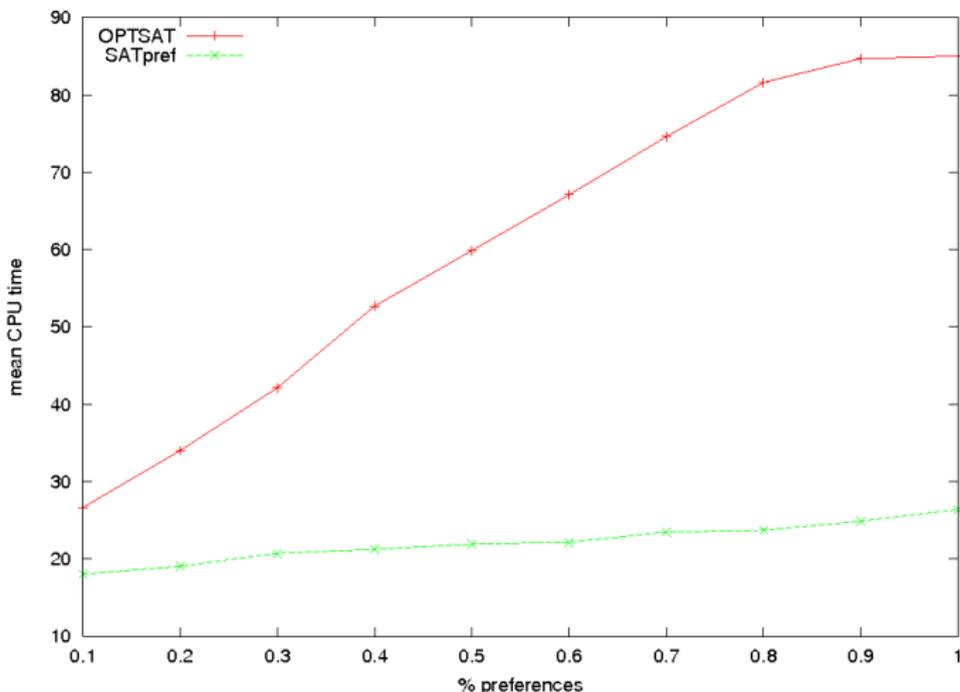


Figure: Degradation of the performances for OPTSAT and SATPREF on the barrel7 instance. All instances are satisfiable.

We have used / are using / would like to use our framework and algorithms in the following applications:

- Planning as Satisfiability with preferences by using
 - 1 OPT-DLL (Giunchiglia and Maratea, AAI 2007)
 - 2 SATPREF
- Preferences in NMR, e.g., (disjunctive) ASP, circumscription.

Current work: Algorithms for finding all optimal solutions

- state-of-the-art: no polynomial space algorithms
- We are working on two algorithms, which extend OPT-DLL and PREF-DLL
 - 1 We follow the preferences, and we compute only optimal solutions by adding, for each computed solution μ , a formula which disables the generation of assignments μ' with $\mu \prec \mu'$
 - 2 We do not follow the any order, and we do not add formulas, but we left the SAT solver to generate assignments and then we check if it is optimal (a call to a SAT oracle with a formula involving starting and preference formulas)

- based on resolution rule: $I \vee C_1$ and $\neg I \vee C_2$ resolve into $C_1 \vee C_2$
- is the “root” of DLL ancestor (DP)
- resolution based theorem provers are usually targeted for first-order logic
- DP variable elimination rule has a number of disadvantages w.r.t. DLL splitting

Local search algorithms

- randomly select an assignment for φ
- then try to minimize the unsatisfied clauses
- can not guarantee completeness

- introduced by (Bryant 1992)
- Boolean functions are represented via directed acyclic graphs
- in the worst case the graph is exponentially larger (in the number of variables)
- some operations on the graph and between graphs are very convenient
- (ordered): introduces a total order on the variables
- highly dependent on the ordering of the variables in the graph

Stalmarck's method

- patented proof method developed by Gunnar Stalmarck (1989)
- it is based on a system for natural deduction
- bread-first backtracking algorithm
- commercial tool Prover and SAT solver Heerhugo are based on this method
- solver Heerhugo can actually deal with more than propositional logic

Diamonds problems

Given a parameter K (number of diamonds), these problems are characterized by an exponentially large (2^K) number of Boolean models T , some of which correspond to satisfying SL-assignments; hard instances with a unique satisfying SL-assignment can be generated.

A second parameter, T (related to the number of edge in each diamond), is used to make T larger, further increasing the difficulty.

Variables range over the reals.

BDD - Example

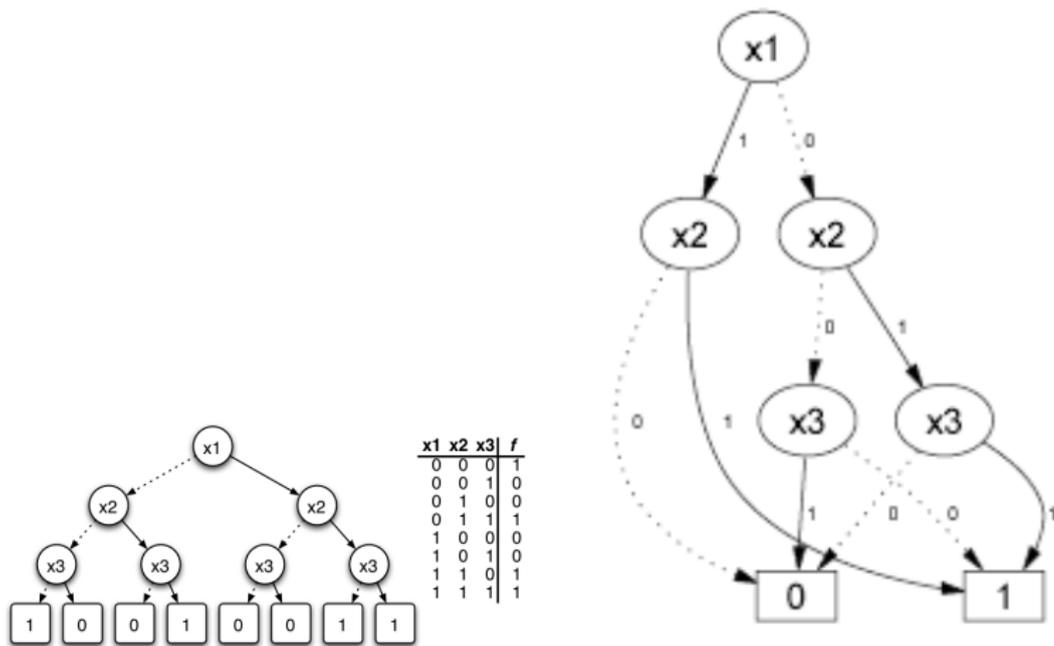


Figure: BDD for the boolean function $f(x_1, x_2, x_3) = \neg x_1 \times \neg x_2 x_3 + x_1 \times x_2 + x_2 \times x_3$. Binary decision tree (left) and binary decision diagram (right).