Propositional Satisfiability (SAT) and SAT-Based Decision Procedures

Marco Maratea

j.w.w. Enrico Giunchiglia and others

Mechanised Reasoning Group

Dipartimento di Informatica, Sistemistica e Telematica - Università di Genova

Motivation

1. Propositional satisfiability (SAT) is one of the most studied fields in AI and CS

2. Very efficient and specialized SAT procedures exist

$\Rightarrow$ use SAT solvers for deciding more expressive logics and formalisms . . .

$\Rightarrow$ . . . reusing most of the work and knowledge available in SAT

SAT: The problem

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

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

Given the clauses $c_1, \ldots, c_m$, a CNF formula is $c_1 \wedge \ldots \wedge c_m$.

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

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 to the propositions in φ such that φ is true?

SAT: Examples

1. $\varphi := \{p, p \vee \neg q, \neg r\}$ has the satisfying assignments $\{p := \mathit{True},\ q := \mathit{True},\ r := \mathit{False}\}$ and $\{p := \mathit{True},\ q := \mathit{False},\ r := \mathit{False}\}$

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.

SAT: Solving methods

- (Ordered) Binary Decision Diagrams (OBDDs)
- Stalmark's method
- Davis-Putnam-Longemann-Loveland (DPLL) algorithm

OBDDs

- boolean functions are represented via directed acyclic graphs
- in the worst case the graph is exponentially (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
- theorem solver Prover and SAT solver Heerhugo are based on this method
- solver Heerhugo can actually deal with more than propositional logic

Agenda

- DPLL
- DPLL-Based Decision Procedures
- Application I: Answer Set Programming (ASP)
- Application II: Separation Logic (SL)
- Ongoing work

DPLL

$\text{SAT}(\varphi)$ **return** $\text{DPLL}(\text{CNF}(\varphi), \emptyset)$;

$\text{DPLL}(\Gamma, S)$
if $\Gamma = \emptyset$ **then return** $True$;
if $\emptyset \in \Gamma$ **then return** $False$;
if $\{l\} \in \Gamma$ **then return** $\text{DPLL}(assign(l, \Gamma), S \cup \{l\})$;
$A :=$ an atom occurring in Γ;
return $\text{DPLL}(assign(A, \Gamma), S \cup \{A\})$ **or**
$\text{DPLL}(assign(\neg A, \Gamma), S \cup \{\neg A\})$.

DPLL-Based decision procedure

Given a formula t in the theory T that can be abstracted/compiled into SAT

$\mathrm{BEY\text{-}SAT}(\mathrm{t})$ **return** $\textsc{dpll}(\mathrm{CNF}(\mathit{Abstract/Compilation}(\mathrm{t})), \emptyset)$;

$\textsc{dpll}(\Gamma, S)$
if $\Gamma = \emptyset$ **then return** $\underline{test(S, t)}$;
if $\emptyset \in \Gamma$ **then return** False;
if $\{l\} \in \Gamma$ **then return** $\textsc{dpll}(assign(l, \Gamma), S \cup \{l\})$;
$A :=$ an atom occurring in Γ;
return $\textsc{dpll}(assign(A, \Gamma), S \cup \{A\})$ **or**
$\textsc{dpll}(assign(\neg A, \Gamma), S \cup \{\neg A\})$.

$test(S, t)$ returns True if S is a "solution" for the formula t, and False, otherwise.

From SAT to BEY-SAT: Discussion

1. $\mathrm{BEY\text{-}SAT(t)}$ returns $True$ iff t has a “solution”

2. $\mathrm{BEY\text{-}SAT(t)}$ can be easily modified in order to compute all the solutions

3. Most SOTA SAT solvers are a (non-recursive) implementation of DPLL

4. Most SOTA SAT solvers are based on “learning” in order to backjump irrelevant nodes while backtracking and avoid the exploration of useless parts of the search tree; it is important that $test(S, t)$ does not return only $False$, but also a “witness” of inconsistency (called *reason*, key point in the algorithm)

5. $\mathrm{BEY\text{-}SAT}$ works in polynomial-space

Application I: Answer Set Programming (ASP)

Answer Set (stable model) Programming is a new programming paradigm proposed by Marek, Truszczynski and Niemela in 1999.

It is a form of declarative programming. It is based on logic rules and on the answer set semantic of Prolog proposed by Gelfond and Lifschitz in 1988.

Basic preliminaries

A *(logic) program* Π is a finite set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n \tag{1}$$

Let P be the set of atoms in Π, $A_0 \in P \cup \{\perp\}$, $\{A_1, \ldots, A_n\} \subseteq$ P. A_0 is the *head*.

$Comp(\Pi)$ consists of formulas of the type

$$A_0 \equiv \bigvee (A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n)$$

for each symbol in $P \cup \{\perp\}$. In the equation, the disjunction extends over all rules (1) in Π with head A_0.

What is an answer set?

Consider first Π in which $m = n$. Let X be a set of atoms.
We say that X is *closed* under Π if for every rule in Π, $A_0 \in X$ whenever $\{A_1, \ldots, A_m\} \subseteq X$.
We say that X is an answer set for Π if X is the smallest set closed under Π.
Consider now the general case $n > m$.
The reduct Π^X of Π related to X is the set of rules

$$A_0 \leftarrow A_1, \ldots, A_m \tag{2}$$

such that $X \cap \{A_{m+1}, \ldots, A_n\} = \emptyset$.
We say that X is an answer set for Π if X is an answer set for Π^X.

Given a (logic) program Π, find if it has a solution is an NP-complete problem.

ASP: Examples

1. Be Π_1: $$\begin{aligned} p &\leftarrow \ not\ q \\ q &\leftarrow \ not\ r \end{aligned}$$

The only AS is $\{q\}$

2. Be Π_2: $$\begin{aligned} p &\leftarrow \ not\ q \\ q &\leftarrow \ not\ p \end{aligned}$$

The ASs are $\{p\}$ and $\{q\}$

3. Be Π_3: $$p \leftarrow \ not\ p$$

Π_3 does not have AS.

Applications of ASP

So far, answer set programming has been used in the following fields:

- planning
- commonsense reasoning
- (bounded) model checking
- VLSI (wire routing)
- ...

ASP-SAT decision procedure

$\text{ASP-SAT}(\Pi)$ **return** $\textsc{dpll}(\text{CNF}(\underline{Comp(\Pi)}), \emptyset)$;

$\textsc{dpll}(\Gamma, S)$
  if $\Gamma = \emptyset$ **then return** $\underline{test(S, \Pi)}$;
  if $\emptyset \in \Gamma$ **then return** $False$;
  if $\{l\} \in \Gamma$ **then return** $\textsc{dpll}(assign(l, \Gamma), S \cup \{l\})$;
  $A :=$ an atom occurring in Γ;
  return $\textsc{dpll}(assign(A, \Gamma), S \cup \{A\})$ **or**
    $\textsc{dpll}(assign(\neg A, \Gamma), S \cup \{\neg A\})$.

$test(S, \Pi)$ returns $True$ if $S \cap P$ is an answer set of Π, and $False$, otherwise.

From SAT to ASP-SAT: Discussion

1. $\mathrm{ASP\text{-}SAT}(\Pi)$ returns $True$ iff Π has an answer set

2. $\mathrm{ASP\text{-}SAT}(\Pi)$ can be easily modified in order to compute all the answer sets of a program Π

3. $test(S, \Pi)$ can fail because of "loops" in the logic program. The reason is extracted from the "loop formulas"

Ex. Π: $p \leftarrow p$, $Comp(\Pi)$ is $p \equiv p$

4. $\mathrm{ASP\text{-}SAT}$ works in polynomial-space

Experimental results: Blocks world

		Standard programs			Extended programs	
#b	#s	SMODELS	ASSAT	CMODELS2	SMODELS	CMODELS2
8	i-1	12.32	0.80	1.19	0.81	0.47
11	i-1	71.78	2.97	4.19	2.97	1.01
8	i	40.87	0.89	2.18	1.56	1.40
11	i	71.42	3.17	4.52	3.41	1.16
8	i+1	23.35	0.96	0.97	4.99	0.31
11	i+1	107.48	3.54	3.33	5.21	0.75

Blocks world: “#b” is the number of blocks.

Experimental results: H.C. complete graphs

	Standard programs				Extended programs	
	SMODELS	ASSAT	DLV	CMODELS2	SMODELS	CMODELS2
np30c	11.70	1.14	22.08	0.69	0.36	0.36
np40c	62.89	41.81	97.96	1.63	2.48	0.87
np50c	219.56	14.51	314.46	3.37	8.39	1.79
np60c	594.46	48.80	770.07	5.81	20.47	3.41
np70c	1323.61	291.60	1679.12	8.22	39.41	5.87
np80c	2354.28	32.51	3407.35	14.20	75.36	9.18
np90c	TIME	779.06	TIME	22.23	122.53	14.19
np100c	TIME	–	TIME	28.63	185.52	20.76
np120c	TIME	–	TIME	53.33	418.15	41.84

Complete graphs. npXc corresponds to a graph with "X" nodes.

Experimental results: FV problems

	SMODELS	ASSAT	DLV	CMODELS2
mutex4	33.92	(0)0.62	840.60	(0)0.68
phi4	0.24	(168)2.98	1.44	TIME
mutex2	0.09	(88)1.78		(0)0.12
mutex3	229.57	MEM		(0)24.16
phi3	2.87	(704)236.91		(57)3.91

Checking requirements in a deterministic automaton.

Experimental results: BMC problems

BMC	SMODELS	CMODELS2	CMODELS2'
dp-10.i-02-b11	382.72	1476.72	442.14
dp-10.s-02-b8	15.24	8.20	14.22
dp-12.s-O2-b9	336.03	65.41	137.34
dp-8.i-O2-b9	8.08	12.62	10.69
dp-8.s-O2-b7	1.19	1.02	2.28
dp-10.i-O2-b12	445.47	3295.72	163.29
dp-10.s-O2-b9	28.87	16.07	15.03
dp-12.s-O2-b10	971.50	209.29	48.73
dp-8.i-O2-b10	5.05	40.01	6.44
dp-8.s-O2-b8	1.76	1.99	2.03

Bounded Model Checking Problems.

Future work on Cmodels2

1. Working on logic programs structure to enhance SAT search (one of the Truszczynski's proposed challenge at NMR'04)

2. Extending the approach to disjunctive logic programs (DLPs).

   DLPs contain rules of the form

   $A_{0,1}; \ldots ; A_{0,r};\ not\ A_{0,r+1}; \ldots ;\ not\ A_{0k} \leftarrow A_1, \ldots , A_m, not\ A_{m+1}, \ldots , not\ A_n$

   Checking if a DLP has an answer set is a Σ_2^p-complete problem.

Application II: Separation Logic (SL)

Decision procedures able to decide quantifier-free first-order theories are becoming increasingly important in Artificial Intelligence and Formal Verification areas.

Several properties of hardware, timed automata, and software can be modeled in quantifier-free first-order theories as well as planning and scheduling problems.

Separation Logic (SL) is one of such decidable quantifier-free first-order theories that allows boolean combination of difference constraints.

Why Separation Logic?

SL seems to be a good compromise between efficiency and expressivity.

It combines propositional atoms with a restricted form of linear arithmetic via the standard boolean connectives.

Many available benchmarks are in SL and a lot of properties of systems and planning/scheduling constraints can be encoded in this logic.

SL: Definitions

Fix a domain of interpretation D for the arithmetic variables (the set of real or the set of integer numbers).

An SL-atom is either a propositional variable or an SL-expression x - y $\leq$ c ($<, >, \geq, =, \neq$ can be (easily) recast in $\leq$), where x and y range on D and c is a numeric constant.

An SL-expression is also called difference constraint.

An SL-literal is an SL-atom or its negation.
An SL-clause is a finite disjunction of SL-literals.
An SL-formula is a finite conjunction of SL-clauses.

Deciding an SL-formula (Is there an SL-assignment to propositional atoms and arithmetic variables, such that the SL-formula ϕ is true?) is an NP-complete problem.

SL-SAT decision procedure

Given a formula ψ in SL,

$$\text{SL-SAT}(\psi) \ \ \textbf{return} \ \textsc{dpll}(\text{CNF}(\underline{\mathit{Abstract}(\psi)}), \emptyset);$$

$\textsc{dpll}(\Gamma, S)$

if $\Gamma = \emptyset$ **then return** $\underline{\mathit{test}(S, \psi)}$;

if $\emptyset \in \Gamma$ **then return** False;

if $\{l\} \in \Gamma$ **then return** $\textsc{dpll}(\mathit{assign}(l, \Gamma), S \cup \{l\})$;

$A :=$ an atom occurring in Γ;

return $\textsc{dpll}(\mathit{assign}(A, \Gamma), S \cup \{A\})$ **or**

$\textsc{dpll}(\mathit{assign}(\neg A, \Gamma), S \cup \{\neg A\})$.

$\mathit{test}(S, \psi)$ returns True if the set of constraints in S is consistent, False otherwise.

From SAT to SL-SAT: Discussion

1. $\text{SL-SAT}(\psi)$ returns $True$ iff ψ has a solution

2. $test(S, \psi)$ can fail because of sets of inconsistent difference constraints in ψ. The reason is extracted from the Bellman-Ford algorithm considering the difference constraints involved in a negative cycle.

3. SL-SAT works in polynomial-space

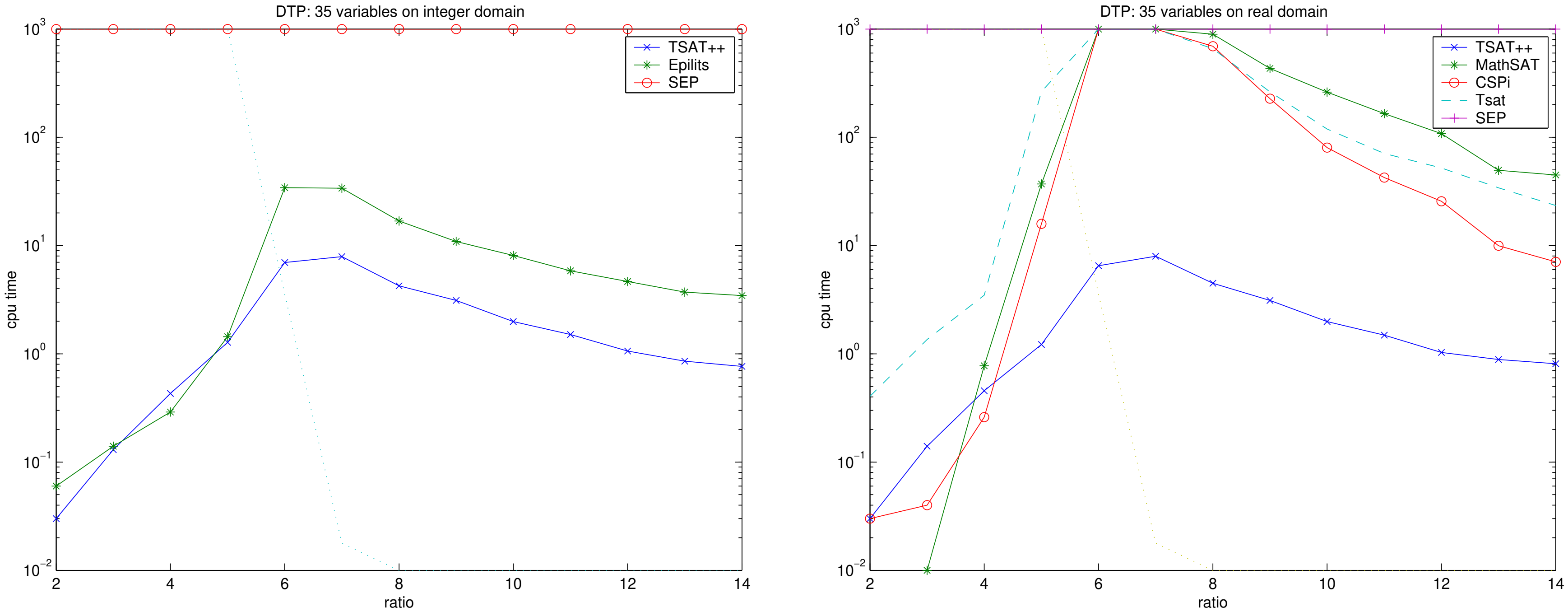
Disjunctive Temporal Problem (DTPs)

These are well-known random problems from the AI community.

DTPs are randomly generated by fixing the number k of expressions x - y $\leq$ c per SL-clause, the number n of arithmetic variables, a positive integer L such that all the constants are taken in $[-L, L]$. Then:

1. the number of clauses m is increased in order to range from satisfiable to unsatisfiable instances from 2*n to 14*n step n,
2. for each tuple of values of the parameters, 100 instances are generated and then given to the solvers, and
3. the median of the CPU time is plotted against the m/n ratio.

TSAT++'s performances (1): DTPs



Evaluation on the DTP on 35 variables. Integer domain (left) and real domain (right). Setting: $k = 2$, $L = 100$.

TSAT++'s performance (2): post-office problems

Instance	SAT?	TSAT++ jp2	MathSAT	SEP
P04-6-P04	NO	0.07	0.36	16.02
P04-7-P04	NO	0.11	0.36	134.21
P04-11-P04	NO	1.01	2.13	TIME
P04-12-P04	YES	0.58	0.91	TIME
P05-10-P05	NO	2.41	5.32	—
P05-11-P05	NO	3.44	9.23	—
P05-12-P05	NO	4.79	22.06	—
P05-13-P05	NO	8.88	54.17	—
P05-14-P05	YES	2.99	11.36	—

Diamonds problems

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

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

Variables range over the reals.

TSAT++'s performance (3): diamonds problems

Instance			Lazy				Eager	
D	S	u?	TSAT++ p2	M.SAT	ICS	CVC	SEP	SEP-m
250	5	NO	0.08	5.40	0.05	MEM	52.20	0.95
250	5	YES	0.21	TIME	150.02	3.26	0.77	288.30
500	5	NO	0.29	21.22	0.11	MEM	742.99	5.92
500	5	YES	1.05	TIME	MEM	6.99	4.85	TIME
1000	5	NO	1.07	–	0.28	MEM	TIME	27.52
1000	5	YES	6.45	–	MEM	15.68	22.53	TIME
2000	5	NO	3.76	–	0.82	MEM	–	–
2000	5	YES	29.90	–	MEM	37.53	–	–

TSAT++'s performance (4): real-world problems from UCLID library

Instance	Lazy		Eager
	TSAT++ p2	ICS	SEP
cache.inv10	0.11	5.29	–
cache.inv12	75.08	53.83	–
dlx1c	TIME	MEM	–
elf.rf8	0.74	2.68	MEM
elf.rf9	13.92	39.24	TIME
ooo.rf7	7.42	16.26	MEM
ooo.rf8	231.80	265.16	TIME
q2.14	230.69	479.65	–

Future work on TSAT++ (1)

From the point of view of the basic research, extending TSAT++'s theory with

- (full) linear arithmetic: general constraints $a_1 * x_1 + a_2 * x_2 + \ldots + a_n * x_n \leq c$
- uninterpreted functions: $f(x_1, x_2, \ldots, x_n)$, used to represent combinatorial ALUs
- arrays: given a and b arrays; i and j indexes, v an element, and the primitives $write(a, i, v)$ and $read(a, i)$ the theory is characterized by the two axioms
  1. $(i = j \to read(write(a, i, v), j) = v) \land$
     $(i \neq j \to read(write(a, i, v), j) = read(a, j))$
  2. $(\forall i \; read(a, i) = read(b, i)) \to a = b$ (for extensionality)

Future work on TSAT++ (2)

On the "applications" side, using TSAT++ as an effective back-end solver for:

- Software Model Checking

  Alessandro Armando, Claudio Castellini and Jacopo Mantovani
  Software Model Checking using Linear Programs
  Accepted to ICFEM 2004

- Planning/Scheduling

  "Activity A1 lasts for 10 units of time at most": $e_1 - s_1 \leq 10$

  "Activity A1 should start before activity A2 finishes": $s_1 \leq e_2$

  "Activity A1 should start before activity A2 finishes, otherwise A3 should start when A2 finishes": $s_1 \leq e_2 \vee s_3 = e_2$

References

http://www.cs.utexas.edu/users/tag/cmodels

http://www.ai.dist.unige.it/Tsat

Yu. Lierler and M. Maratea - Cmodels2: SAT-Based Answer Set Solvers Extended to Non-tight Programs. In Proc. LPNMR 2004

E. Giunchiglia, Yu. Lierler and M. Maratea - SAT-Based Answer Set Programming. In Proc. AAAI 2004

A. Armando, C. Castellini, E. Giunchiglia and M. Maratea - A SAT-Based Decision Procedure for the Boolean Combination of Difference Constraints. Accepted to SAT 2004

A. Armando, C. Castellini, E. Giunchiglia, M. Idini and M. Maratea - TSAT++: An Open Reasoning Platform for Satisfiability Modulo Theory. PDPAR 04

Ongoing work on SAT-Based DPs (1)

There is some interesting research related to SAT, namely

1. Max-SAT: Given an unsatisfiable instance, how many clauses can be satisfied at most (at the same time)?

2. Max(Min)-One: Given an satisfiable instance, find the satisfying assignment with the maximum (minimum) number of variables assigned to $True$

3. Minimum(Minimal) Unsatisfiable Core (UC): Given an unsatisfiable instance, find the minimum (in the number of clauses) or minimal (under subset inclusion) set of clauses that are unsatisfiable

Note that 1. and 3. are not the "symmetric" problem.
Ex. $\{p \wedge q, p, \neg p\}$, Max-SAT set is $\{p \wedge q, p\}$ while the UC is $\{p, \neg p\}$

Ongoing work on SAT-Based DPs (2)

These problems have applications in routing problems, planning, (unbounded) model checking, correcting the minimum amount of inconsistent knowledge.

Till now, methods for resolving these problems have been focused on

- branch-and-bound algorithm
- extensions of the DPLL algorithm to reason with constraints (the first two problems can be seen as a cardinality constraint)
- modification the DPLL algorithm to deal with this problems (especially for 3.)

and all have to look at the entire search space.

Our idea: A general framework to deal with these problems.

Ongoing work on SAT-Based DPs: Our approach

Let focus for the moment on the first two problems, Max-SAT and Max-One.

Both problems can be expressed via an optimization function of the type $max \sum_i x_i$ where the x_is are (a subset of) the variables in the formula.

A cardinality constraints $\sum_i x_i$ can be encoded (via half (or full) adder, specialized encoding) in a propositional formula $\mathrm{ENC}(x_i, s_i, b_i)$ where s_i are some added variables and $b_0, \ldots, b_m$ is the binary representation of the cardinality constraint.

Given a CNF formula $\phi(x_i)$ the idea is to build the extended CNF formula $\phi(x_i) \wedge \mathrm{CNF}(\mathrm{ENC}(x_i, s_i, b_i))$ and than search in its search space guiding the search with the b_i bits from the MSB to the LSB.

Modification in the DPLL algorithm: OPT-SAT

$\mathrm{OPT\text{-}SAT}(\phi)$ **return** $\textsc{dpll}(\phi \wedge \mathrm{CNF}(\mathrm{ENC}(x_i, s_i, b_i)), \emptyset)$;

$\textsc{dpll}(\Gamma, S)$
  if $\Gamma = \emptyset$ **then return** $\underline{test(S, \phi)}$;
  if $\emptyset \in \Gamma$ **then return** $False$;
  if $\{l\} \in \Gamma$ **then return** $\textsc{dpll}(assign(l, \Gamma), S \cup \{l\})$;
  $A :=$ an atom occurring in Γ, <u>preferentially</u> and <u>in order</u> on b_i;
  return $\textsc{dpll}(assign(A, \Gamma), S \cup \{A\})$ **or**
      $\textsc{dpll}(assign(\neg A, \Gamma), S \cup \{\neg A\})$.

$test(S, \phi)$ does not check the solution; it can be used for finding all the solutions with the same "rank" (for ex. Max-One), otherwise return $True$

(Minimum/Minimal) Unsatisfiable cores

This problem turned out to be the most tricky.

Probably there is not a direct characterization as optimization function in terms of cardinality/linear constraint. An idea is that the problem can be expressed using a QBF formula of the type $min_W\ (\neg\exists W\forall(X - W)\neg\phi(X))$ where X is the set of variables and W its subset called "clause selectors". This open the way to the use of QBF solver, another technology that we have "in house".

Nevertheless, we can also attack the weighted versions of the Max-SAT and Max-One problems directly extending the framework before.