Mathematical Programming: Modelling and Applications

Sonia Cafieri

LIX, École Polytechnique

cafieri@lix.polytechnique.fr

November 2009

Boolean variables, Literals and Clauses

Definition

A *boolean variable* x is a variable that can assume only two values 0 and 1.

0 is for *false*, 1 for *true*.

The *negation $\bar{x}$ of a boolean variable* x is the variable that assumes the value $1 - x$.

Definition

A *literal l* is a variable x or its negation $\bar{x}$.

Let X be a set of boolean variables. For every $x \in X$ there are 2 literals over x, namely x itself and $\bar{x}$.

Definition

A *clause* over a set of boolean variables is a disjunction of literals.

Example:
if we have 2 variables x_1, x_2, we create a clause by using the *or* operator: $x_1 \vee \bar{x}_2$.

CNF formula

Definition

A *boolean formula in Conjunctive Normal Form* (CNF) Φ is a conjunction of clauses:

$$\Phi = \bigwedge_{i=1}^{m} C_i, \quad \text{where } C_i \text{ are clauses, } \forall i \in \{1, .., m\}.$$

Example:

suppose we have 3 variables: x_1, x_2, x_3

we can create some clauses by using the *or* operator:

$$(x_1 \vee \bar{x}_2), (\bar{x}_1 \vee x_2), (\bar{x}_2 \vee \bar{x}_3), (x_1 \vee x_3), (\bar{x}_1 \vee \bar{x}_2), (\bar{x}_1 \vee x_2)$$

then we can join the clauses by using the *and* operator:

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2)$$

Definition

The *size* of a CNF formula is the sum of the length of all its clauses.

Assigment of values

Definition

An assigment of values to the set X of variables of a boolean formula is called a *truth assignment*.

An assignment of truth values to the propositional variables:

- *satisfies* a literal x_i if x_i takes the value 1
- *satisfies* a literal $\bar{x}_i$ if x_i takes the value 0
- *satisfies* a clause if it satisfies at least one literal of the clause
- *satisfies* a CNF formula if it satisfies all the clauses of the formula.

An assignmnent for a CNF formula Φ is *complete* if all the variables occurring in Φ have been assigned, otherwise it is *partial*.

The SAT problem

Definition

Given a boolean formula in Conjunctive Normal Form, consisting of a conjunction of m clauses C_i, each of which with 2 literals, the *2-Satisfiability Problem*, also called **2-SAT**, is the problem of deciding if there is a truth assignment to the literals such that each clause is satisfied (i.e. its value is 1).

Depending on whether this is possible or not, we say that the formula is satisfiable or unsatisfiable.

Each instance of the 2-SAT problem is a 2CNF formula.

The MAX2SAT problem

Definition

decision version

Given a boolean formula in Conjunctive Normal Form, consisting of a conjunction of m clauses C_i, each of which with 2 literals, and given an integer k, the *Maximum 2-Satisfiability problem*, also called **MAX2SAT**, is the problem of deciding if there is a truth assignment to the literals such that *satisfies at least k clauses*.

Definition

optimization version

Given a boolean formula in Conjunctive Normal Form, consisting of a conjunction of m clauses C_i, each of which with 2 literals, the *Maximum 2-Satisfiability problem*, also called **MAX2SAT**, is to find a truth assignment to the literals that *maximizes the number of satisfied clauses*.

The MAXSAT problem

The natural generalization of MAX2SAT is the MAXSAT problem.

Definition

The *Maximum Satisfiability problem* (**MAXSAT**) asks for the maximum number of clauses which can be satisfied by any assignment.
The clauses have not a limit on the number of literals.

Solution methods

Two classes of methods are used to solve SAT and MAXSAT problems: *exact methods* and *approximate methods*.

Exact methods

- guarantee finding an optimal solution for an instance or, if there is no solution, proving its unsatisfiability
- perform exhaustive searches
- do not check every possible configuration (they usually use strategies to speed up the search)
- their performance can be limited for large instances
- are generally based on the exploration of a binary search tree, building incrementally truth assignments.

Approximate methods

- do not guarantee finding an optimal solution
- are quickly in finding a solution (if they find it)
- are mainly based on local search and evolutionary algorithms.

RandomWalk algorithm

An approximate method.

Basic algorithm:

- Randomly choose a truth assignment for the variables.
- If the current assignment satisfies the formula (for SAT) or the required number of clauses (for MAXSAT), then terminate the procedure.
- Else randomly pick a clause which is unsatisfied by the current assignment and flip a variable within such clause.

Note: GSAT, WalkSAT, ... , are variants of the RandomWalk algorithm.

A RandomWalk algorithm for MAX2SAT

Suppose we have MAX2SAT – decision version. We aim to satisfy k clauses.

```
Input a 2CNF instance and a positive integer $k$

randomly assign a 0/1 value to all variables
termination := 0
while (termination = 0) do
   replace the variables in the formula with the selected assignment
   if (at least $k$ clauses are satisfied) then
      termination := 1
   else
      choose randomly an unsatisfied clause
      choose randomly a variable within the selected clause and change its value
   endif
   if (iteration limit reached) then
      termination := 1
   endif
endwhile
```

The output is the truth assignment that satisfies k clauses (if the solution is found).

A RandomWalk algorithm for MAX2SAT

Some observation.

- The algorithm does not guarantee finding an optimal solution.
- If a solution is found, it is usually quickly found.
- At each step, the clause to be changed can be randomly chosen.
- The fact that a clause is unsatisfied means that at least one of the variables in the clause must be flipped in order to reach a global solution.
- In some algorithms, the variable to be flipped is chosen from the selected unsatisfied clause by some greedy heuristic.
- A number of such heuristics have been studied. For ex., some algorithms take into account the number of currently satisfied clauses that would become unsatisfied if a variable were to be flipped.

Exercise

Write an AMPL script implementing the RandomWalk algorithm for the MAX2SAT problem.

Use the decision version of the problem, requiring to satisfy k clauses.

- Use a maximum number of allowed iterations for the main loop.
- Among the unsatisfied clauses, you can select, for ex., the first unsatisfied clause.
- When a variable within a clause is chosen to be changed, store the information on the pair (clause, variable) that have been chosen and use this information to avoid using the same pair during the next iterations.

AMPL scripts

AMPL scripts can contain any AMPL command and may include programming language constructs like `for`, `repeat` and `if` to repeat statements and perform them conditionally [Fourer, Gay, Kernighan, AMPL, 2nd edition].

- Use programming language constructs like `for`, `repeat` and `if`.
- To assign random values, you can use the AMPL function `Uniform01()` that returns a random number uniformly distributed between 0 and 1.

AMPL scripts

- `for`: executes commands for each member of a set.
  Example:

```
for {i in 1..4} {
  let v[i] := i;
}
```

- `repeat`: continues iterating as long as a logical condition is satisfied.
  Example:

```
repeat while (i <= 4) {
  let v[i] := i;
}
```

- `if`: evaluates a condition and executes some specified command if the condition is true.
  Example:

```
if (i <> 4) then {
  let v[i] := i;
}
```

An instance

$$m = 9 \text{ clauses}$$
$$n = 3 \text{ variables}$$

$$
\begin{gathered}
(x_1 \vee \bar{x}_2) \wedge \\
(\bar{x}_1 \vee x_2) \wedge \\
(\bar{x}_2 \vee \bar{x}_3) \wedge \\
(x_1 \vee x_3) \wedge \\
(\bar{x}_1 \vee \bar{x}_2) \wedge \\
(x_1 \vee x_2) \wedge \\
(x_2 \vee x_3) \wedge \\
(x_2 \vee \bar{x}_3) \wedge \\
(x_1 \vee \bar{x}_3)
\end{gathered}
$$

We want to satisfy $m - 1 = 8$ clauses.

An instance

Use the function φ defined as:

$$\begin{aligned}\varphi(i,j) &= 1 && \text{if } x_j \text{ appears in clause } i\\ &= -1 && \text{if } \bar{x}_j \text{ appears in clause } i\\ &= 0 && \text{otherwise}\end{aligned}$$

This can be useful to encode an instance.

It can also be used to test if a clause is satisfied by a selected assignment.

An instance - AMPL data file

```
param m := 9;
param n := 3;
param phi :=
1 1   1
1 2  -1
1 3   0
2 1  -1
2 2   1
2 3   0
3 1   0
3 2  -1
3 3  -1
4 1   1
4 2   0
4 3   1
5 1  -1
5 2  -1
5 3   0
6 1  -1
6 2   1
6 3   0
7 1   0
7 2   1
7 3   1
8 1   0
8 2   1
8 3  -1
9 1   1
9 2   0
9 3  -1
;
```