DM841

Discrete Optimization

Satisfiability
Incomplete Algorithms

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Maximum Weighted Satisfiability

Notation:

- 0-1 variables x_j, $j \in N = \{1, 2, \dots, n\}$,
- clauses C_i, $i \in M = \{1, 2, \dots, m\}$, and weights w_i (≥ 0), $i \in M$
- $\bar{x}_j = 1 - x_j$
- $L = \bigcup_{j \in N} \{x_j, \bar{x}_j\}$ set of literals
- $C_i \subseteq L$ for $i \in M$ (e.g., $C_i = \{x_1, \bar{x}_3, x_8\}$).
- Task: $\max_{x \in \{0,1\}^n} \sum \{w_i \mid i \in M \text{ and } C_i \text{ is satisfied in } x\}$

1. devise preprocessing rules, ie, polynomial time simplification rules
2. design one or more construction heuristics for the problem
3. design one or more local search for the problem

Pre-processing

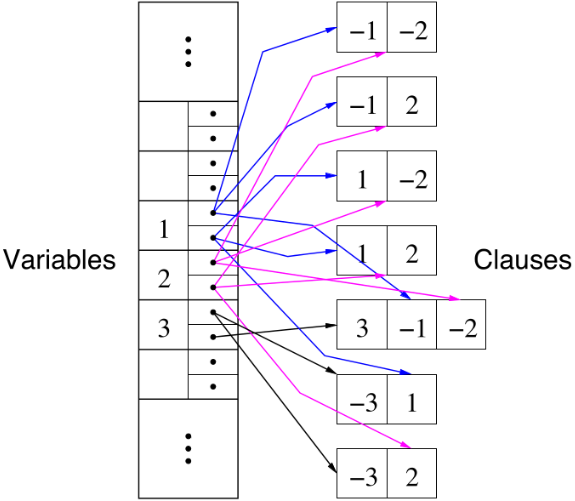
Pre-processing rules: low polynimial time procedures to decrease the size of the problem instance.

Typically applied repeatedly until fixed point when no rule is effective anymore.

Examples in SAT

1. eliminate duplicate literals
2. eliminate tautologies: $x_1 \vee \neg x_1 \ldots$
3. eliminate subsumed clauses
4. eliminate clauses with pure literals
5. eliminate unit clauses
6. unit resolution/propagation
7. probing (existential quantification for each variable)

Simple data structure for unit propagation



Local Search: Decision vs Minimization

LS-Decision(π)
input: problem instance $\pi \in \Pi$
output: solution $x \in S'(\pi)$ **or** $\emptyset$

$(x) := \texttt{init}(\pi)$

while not $\texttt{terminate}(\pi, \texttt{x})$ **do**
    $(x) := \texttt{step}(\pi, \texttt{x})$

if $x \in S'(\pi)$ **then**
    return x
else
    return $\emptyset$

LS-Minimization(π')
input: problem instance $\pi' \in \Pi'$
output: solution $x \in S'(\pi')$ **or** $\emptyset$

$(x) := \texttt{init}(\pi')$;
$x_b := x$;
while not $\texttt{terminate}(\pi', \texttt{x})$ **do**
    $(x) := \texttt{step}(\pi', \texttt{x})$;
    if $f(\pi', x) < f(\pi', x_b)$ **then**
        $x_b := x$;

if $x_b \in S'(\pi')$ **then**
    return x_b
else
    return $\emptyset$

- Assignment: $x \in \{0, 1\}^n$
- Evaluation function: $f(x) = \#$ unsatisfied clauses (assume $w_i = 1$)
- Neighborhood: one-flip
- Step rule: best neighbor

Naive approach: exahustive neighborhood examination in $O(nmk)$ (k size of largest C_i)
A better approach:

- $C(x_j) = \{i \in M \mid x_j \in C_i\}$ (i.e., clauses dependent on x_j)
- $L(x_j) = \{\ell \in N \mid \exists i \in M \text{ with } x_\ell \in C_i \text{ and } x_j \in C_i\}$
- $f(\mathrm{x}) = \#$ unsatisfied clauses
- $\Delta(x_j) = f(\mathrm{x}') - f(\mathrm{x}), \qquad \mathrm{x}' = \delta_{1E}^{x_j}(\mathrm{x})$ (aka score of x_j)

Initialize:

- compute f, score of each variable, and list unsat clauses in $O(mk)$
- init $C(x_j)$ for all variables

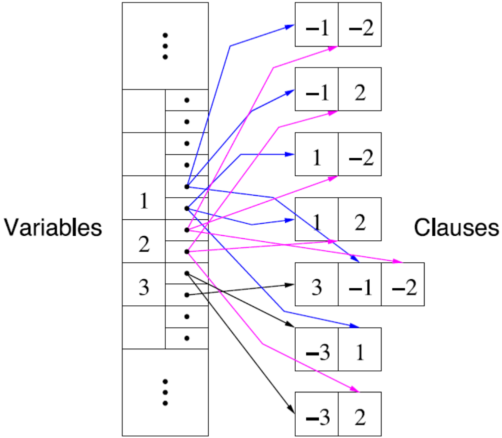
Examine Neighborhood

- choose the var with best score

Update:

- change the score of variables affected, that is, look in $C(\cdot)$ $O(mk)$

$C(x_j)$ Data Structure



Even better approach (though same asymptotic complexity):
$\rightsquigarrow$ after the flip of x_j only the score of variables in $L(x_j)$ that critically depend on x_j actually changes

- Clause C_i is critically satisfied by a variable x_j in x iff:
    - x_j is in C_i
    - C_i is satisfied in x and flipping x_j makes C_i unsatisfied
      (e.g., $1 \vee 0 \vee 0$ but not $1 \vee 1 \vee 0$)

  Keep a list of such clauses for each var

- x_j is critically dependent on x_ℓ under x iff:
  there exists $C_i \in C(x_j) \cap C(x_\ell)$ and such that flipping x_j:
    - C_i changes from satisfied to not satisfied or viceversa
    - C_i changes from satisfied to critically satisfied by x_ℓ or viceversa

Initialize:

- compute score of variables;
- init $C(x_j)$ for all variables
- init status criticality for each clause (ie, count # of ones per clause)

Update:

change sign to score of x_j
for all C_i in $C(x_j)$ where critically dependent vars are **do**
    for all $x_\ell \in C_i$ **do**
        update score x_ℓ depending on its critical status before flipping x_j

Summary

1. Local Search for SAT