DM811
Heuristics for Combinatorial Optimization

Lecture 13
Examples

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

1. Recap.

2. Other Combinatorial Optimization Problems
   GCP
   SAT
   p-median Problem
   Covering and Partitioning

Summary: Local Search Algorithms
(as in [Hoos, Stützle, 2005])

For given problem instance π:

1. search space S_π
2. neighborhood relation $\mathcal{N}_\pi \subseteq S_\pi \times S_\pi$
3. evaluation function $f_\pi : S \to \mathbf{R}$
4. set of memory states M_π
5. initialization function $\texttt{init} : \emptyset \to S_\pi \times M_\pi)$
6. step function $\texttt{step} : S_\pi \times M_\pi \to S_\pi \times M_\pi$
7. termination predicate $\texttt{terminate} : S_\pi \times M_\pi \to \{\top, \bot\}$

After implementation and test of the above components, improvements in efficiency (ie, computation time) can be achieved by:

A. fast delta evaluation
B. neighborhood pruning
C. clever use of data structures

Improvements in quality can be achieved by:

D. application of a metaheuristic
E. definition of a larger neighborhood

Outline

Approach: K-fixed / complete / improper

Local Search

- Solution representation: var{int} a[|V|](1..K)
- Evaluation function: conflicting edges
- Neighborhood: one-exchange
- Pivoting rule: best neighbor

Naive approach: $O(n^2k)$
Neighborhood examination
for all $v \in V$ **do**
  for all $k \in 1..k$ **do**
    compute $\Delta(v, k)$

Better approach:

- V^c set of vertices involved in a conflict
- adj_in_class[n][K] stores number of vertices adjacent in each color class

Initialize:

- compute adj_in_class[n][K] and V^c in $O(n^2)$

Neighborhood examination:

for all $v \in V^c$ **do**
  for all $k \in 1..k$ **do**
    compute $\Delta(v, k) = adj\_in\_class[v][k] - adj\_in\_class[v][a(v)]$

Update:

- change adj_in_class[n][K] and V^c in $O(n^2)$

SAT

- n 0-1 variables x_j, $j \in N = \{1, 2, ..., n\}$,
- m clauses C_i, $i \in M$, and weights w_i (≥ 0), $i \in M = \{1, 2, ..., m\}$
- $\max_{\mathbf{a} \in \{0,1\}^n} \sum \{w_i | i \in M$ and C_i is satisfied in $\mathbf{a}\}$
- $\bar{x}_j = 1 - x_j$
- $L = \bigcup_{j \in N} \{x_j, \bar{x_j}\}$ set of literals
- $C_i \subseteq L$ for $i \in M$ (e.g., $C_i = \{x_1, \bar{x_3}, x_8\}$).

Let's take the case $w_j = 1$ for all $j \in N$

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

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

- $C(x_j) = \{i \in M | x_j \in C_i\}$ (i.e., clauses dependent on x_j)
- $L(x_j) = \{l \in N | \exists i \in M$ with $x_l \in C_i$ and $x_j \in C_i\}$
- $f(a(x_j)) = \#$ unsatisfied clauses
- Score of x_j: $\Delta(x_j) = f(a(x_j)) - f(1 - a(x_j))$

Initialize:

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

Examine

- choose the var with best score

Update:

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

Even better approach:
⇝ 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 $\mathbf{a}$ iff:
  - x_j is in C_i
  - C_i is satisfied in $\mathbf{a}$ and flipping x_j makes C_i unsatisfied (e.g., $1 \vee 0 \vee 0$ but not $1 \vee 1 \vee 0$)

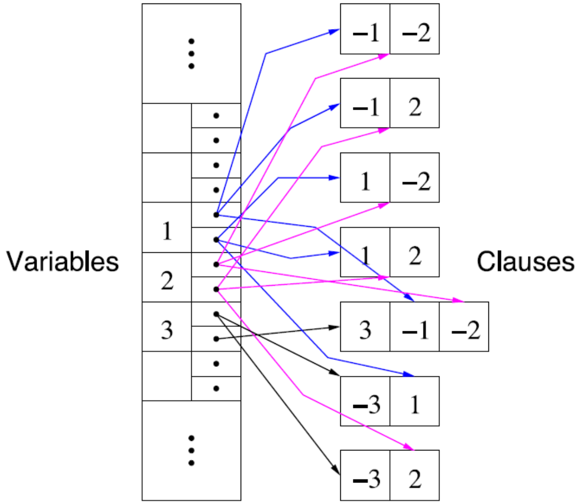
  Keep a list of such clauses for each var
- x_j is critically dependent on x_l under $\mathbf{a}$ iff: there exists $C_i \in C(x_j) \cap C(x_l)$ and such that flipping x_j:
  - C_j changes satisfaction status
  - C_j changes satisfied /critically satisfied status

Initialize:

- compute score of variables;
- init $C(x_j)$ for all variables
- init status criticality for all clauses

Update:
change sign to score of x_j
for all C_i in $C(x_j)$ **do**
  for all $x_l \in C_i$ **do**
    update score x_l depending on its critical status before flipping x_j

Data Structure



The p-median Problem

Given:
a set F of locations of m facilities
a set U of locations for n users
a distance matrix $D = [d_{ij}] \in \mathbf{R}^{n \times m}$
Task: Select p locations of F where to install facilities such that the sum of the distances of each user to its closest installed facility is minimized, *i.e.*,

$$\min \left\{ \sum_{i \in U} \min_{j \in J} d_{ij} \mid J \subseteq F \text{ and } |J| = p \right\}$$

Graph Problems

Max Independent Set (aka, stable set problem or vertex packing problem)

Given: an undirected graph $G(V, E)$ and a non-negative weight function ω on V ($\omega : V \to \mathbf{R}$)
Task: A largest weight independent set of vertices, i.e., a subset $V' \subseteq V$ such that no two vertices in V' are joined by an edge in E.

Maximum Clique

Given: an undirected graph $G(V, E)$
Task: A maximum cardinality clique, i.e., a subset $V' \subseteq V$ such that every two vertices in V' are joined by an edge in E

Vertex Cover

Given: an undirected graph $G(V, E)$ and a non-negative weight function ω on V ($\omega : V \to \mathbf{R}$)
Task: A smallest weight vertex cover, i.e., a subset $V' \subseteq V$ such that each edge of G has at least one endpoint in V'.

Compare with Dominating Set

Set Problems

Set Covering

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \\ & x_j \in \{0, 1\} \end{aligned}$$

Set Partitioning

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j = 1 \quad \forall i \\ & x_j \in \{0, 1\} \end{aligned}$$

Set Packing

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq 1 \quad \forall i \\ & x_j \in \{0, 1\} \end{aligned}$$

The independent set problem is equivalent to the set packing.
Vertex cover problem is a strict special case of set covering.