DM841
Discrete Optimization

Exercises

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

To activate the `Observer` from command line with `--main::observer 1`:

1. declare the parameter from command line:

```
Parameter<unsigned> observer("observer", "Attach the observers", main_parameters);
```

2. declare the observer:

```
RunnerObserver<BDS_Input, BDS_State, FlipOrSwap,int> ob(observer,0);
```

3. attach the observer to the runner:

```
if (observer.IsSet())
    bds_sa.AttachObserver(ob);
```

You can print the number of iterations made by the runner with

```
runner.Iteration()
```

this can be useful for assessing speedups.

It is possible to set the time limit from command line. In order to do this you have to move the declaration of the Solver, for example:

```
SimpleLocalSearch<XYZ_Input, XYZ_Output, XYZ_State, int> XYZ_solver(in, XYZ_sm,
        XYZ_om, "XYZ solver");
```

before the second call of `CommandLineParameters::Parse()`

Efficiency and Effectiveness

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

A. fast incremental evaluation (ie, delta evaluation)

B. neighborhood pruning

C. clever use of data structures

Improvements in effectiveness, ie, quality, can be achieved by:

D. application of a metaheuristic

E. definition of a larger neighborhood

Outline

1. Examples
   SAT

2. Examples

Outline

1. Examples
   SAT

2. Examples

MAX-SAT

Notation:

- n 0-1 variables x_j, $j \in N = \{1, 2, ..., n\}$,
- m clauses C_i, $i \in M$, and weights w_i (≥ 0), $i \in M = \{1, 2, \ldots, m\}$
- $\max_{\mathbf{a} \in \{0,1\}^n} \sum \{w_i \mid i \in M \text{ and } C_i \text{ is satisfied in } \mathbf{a}\}$
- $\bar{x}_j = 1 - x_j$
- $L = \bigcup_{j \in N} \{x_j, \bar{x}_j\}$ 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 $w_i = 1$ for all $i \in M$

- 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:

- $C(x_j) = \{i \in M \mid x_j \in C_i\}$ (i.e., clauses dependent on x_j)
- $L(x_j) = \{l \in N \mid \exists i \in M \text{ with } x_l \in C_i \text{ and } x_j \in C_i\}$
- $f(\mathbf{a}) = \#$ unsatisfied clauses
- $\Delta(x_j) = f(\mathbf{a}) - f(\mathbf{a}'), \mathbf{a}' = \delta_{1E}^{x_j}(\mathbf{a})$ (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

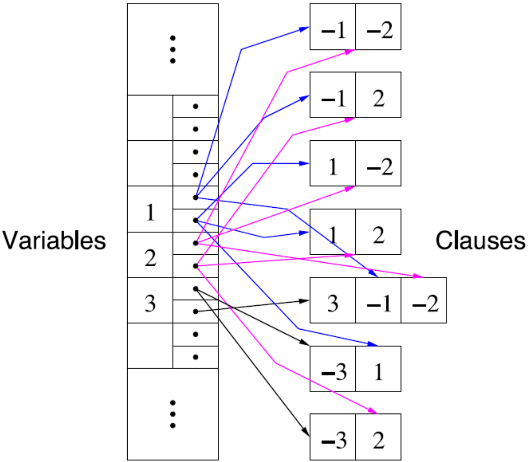
Examine Neighborhood

- choose the var with best score

Update:

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

$C(x_j)$ Data Structure



Even better approach (though same asymptotic complexity):
$\leadsto$ 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 **a** iff:
  - x_j is in C_i
  - C_i is satisfied in **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 **a** iff:
  there exists $C_i \in C(x_j) \cap C(x_l)$ and such that flipping x_j:
  - C_i changes satisfaction status
  - C_i changes satisfied /critically satisfied status

Initialize:

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

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

Outline

1. Examples
   SAT

2. Examples

Examples

- Permutations
  - TSP
  - SMWTP
- Assignments
  - SAT
  - Coloring
  - Parallel machines
- Sets
  - Max Weighted Independent Set
  - Steiner tree

Single Machine Total Weighted Tardiness

Given: a set of n jobs $\{J_1, \ldots, J_n\}$ to be processed on a single machine and for each job J_i a processing time p_i, a weight w_i and a due date d_i.

Task: Find a schedule that minimizes
the total weighted tardiness $\sum_{i=1}^{n} w_i \cdot T_i$
where $T_i = \max\{C_i - d_i, 0\}$ (C_i completion time of job J_i)

Example:

Job	J_1	J_2	J_3	J_4	J_5	J_6
Processing Time	3	2	2	3	4	3
Due date	6	13	4	9	7	17
Weight	2	3	1	5	1	2

Sequence $\phi = J_3, J_1, J_5, J_4, J_1, J_6$

Job	J_3	J_1	J_5	J_4	J_2	J_6
C_i	2	5	9	12	14	17
T_i	0	0	2	3	1	0
$w_i \cdot T_i$	0	0	2	15	3	0

The Max Independent Set Problem

Also called "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.

Related Problems:

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'.

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

Graph Partitioning

Input: A graph $G = (V, E)$, weights $w(v) \in Z^+$ for each $v \in V$ and $l(e) \in Z^+$ for each $e \in E$.
Task: Find a partition of V into disjoint sets $V_1, V_2, \ldots, V_m$ such that $\sum_{v \in V_i} w(v) \leq K$ for $1 \leq i \leq m$ and such that if $E' \subseteq E$ is the set of edges that have their two endpoints in two different sets V_i, then $\sum_{e \in E'} l(e)$ is minimal.

Consider the specific case of graph bipartitioning, that is, the case $|V| = 2n$ and $K = n$ and $w(v) = 1, \forall v \in V$.

Example: Scheduling in Parallel Machines

Total Weighted Completion Time on Unrelated Parallel Machines Problem

Input: A set of jobs J to be processed on a set of parallel machines M. Each job $j \in J$ has a weight w_j and processing time p_{ij} that depends on the machine $i \in M$ on which it is processed.

Task: Find a schedule of the jobs on the machines such that the sum of weighted completion time of the jobs is minimal.

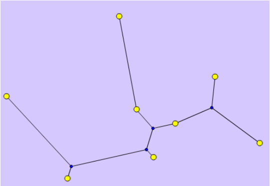
Example: Steiner Tree

Steiner Tree Problem

Input: A graph $G = (V, E)$, a weight function $\omega : E \mapsto \mathbf{N}$, and a subset $U \subseteq V$.

Task: Find a Steiner tree, that is, a subtree $T = (V_T, E_T)$ of G that includes all the vertices of U and such that the sum of the weights of the edges in the subtree is minimal.

Vertices in U are the special vertices and vertices in $S = V \setminus U$ are Steiner vertices.



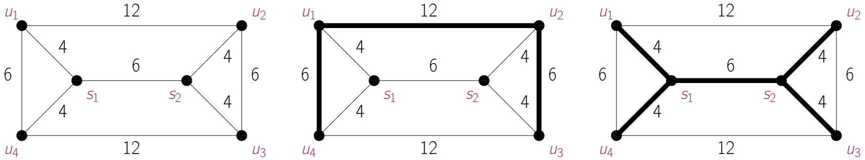


Figure : Vertices u_1, u_2, u_3, u_4 belong to the set U of special vertices to be covered and vertices s_1, s_2 belong to the set S of Steiner vertices. The Steiner tree in the second graph has cost 24 while the one in the third graph has cost 22.

1. Design one or more local search algorithms for the Steiner tree problem. In particular, define the solution representation and the neighborhood function.
2. Provide an analysis of the computational cost of the basic operations in the local search algorithms designed at the previous point. In particular, consider the size of the neighborhood, and the cost of evaluating a neighbor.