DM811

Heuristics for Combinatorial Optimization

Efficient Local Search

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Course Overview

- ✔ Combinatorial Optimization, Methods and Models
- ✔ CH and LS: overview
- ✔ Working Environment and Solver Systems
- ~ Methods for the Analysis of Experimental Results
- ✔ Construction Heuristics
- ✔ Local Search: Components, Basic Algorithms
- Efficient Local Search: Incremental Updates and Neighborhood Pruning
- Local Search: Neighborhoods and Search Landscape
- ~ Stochastic Local Search & Metaheuristics
- Configuration Tools: F-race
- Very Large Scale Neighborhoods

Examples: GCP, CSP, TSP, SAT, MaxIndSet, SMTWP, Steiner Tree, Unrelated Parallel Machines, p-median, set covering, QAP, ...

Outline

1. Efficient Local Search

2. Examples
    - SAT
    - TSP
    - SMTWTP

Outline

1. Efficient Local Search

2. Examples
    - SAT
    - TSP
    - SMTWTP

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

For given problem instance π:

1. search space S_π
2. evaluation function $f_\pi : S \to \mathbf{R}$
3. neighborhood relation $\mathcal{N}_\pi \subseteq S_\pi \times S_\pi$
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 $\texttt{terminate} : S_\pi \times M_\pi \to \{\top, \bot\}$

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

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 $w_j = 1$ 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:

- $C(x_j) = \{i \in M \mid x_j \in C_i\}$ (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

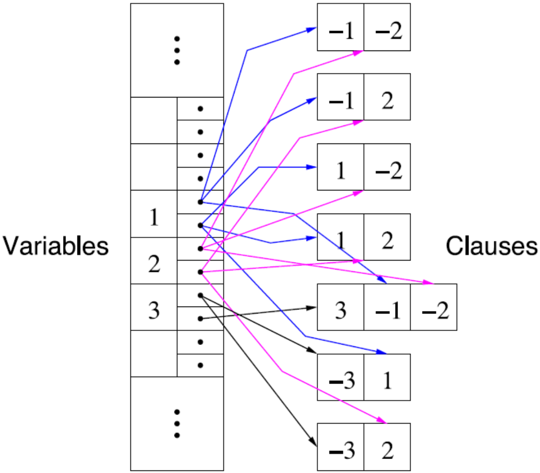
Examine Neighborhood

- 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 $\mathbf{a}$ 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_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

TSP

Efficient implementations of 2-opt, 2H-opt and 3-opt local search.

A. Delta evaluation already in $O(1)$

B. Fixed radius search + DLB

C. Data structures

Details at black board and references [Bentley, 1992; Johnson and McGeoch, 2002; Applegate et al., 2006]

Local Search for the Traveling Salesman Problem

- k-exchange heuristics
  - 2-opt
  - 2.5-opt
  - Or-opt
  - 3-opt
- complex neighborhoods
  - Lin-Kernighan
  - Helsgaun's Lin-Kernighan
  - Dynasearch
  - ejection chains approach

Implementations exploit speed-up techniques

1. neighborhood pruning: fixed radius nearest neighborhood search
2. neighborhood lists: restrict exchanges to most interesting candidates
3. don't look bits: focus perturbative search to "interesting" part
4. sophisticated data structures

Implementation examples by Stützle:
http://www.sls-book.net/implementations.html

TSP data structures

Tour representation:

- determine pos of v in π
- determine succ and prec
- check whether u_k is visited between u_i and u_j
- execute a k-exchange (reversal)

Possible choices:

- $|V| < 1.000$ array for π and π^{-1}
- $|V| < 1.000.000$ two level tree
- $|V| > 1.000.000$ splay tree

Moreover static data structure:

- priority lists
- k-d trees

Look at implementation of local search for TSP by T. Stützle:

File: http://www.imada.sdu.dk/~marco/DM811/Resources/ls.c

```
two_opt_b(tour); % best improvement, no speedup
two_opt_f(tour); % first improvement, no speedup
two_opt_best(tour); % first improvement including speed-ups (dlbs, fixed radius near
      neighbour searches, neughbourhood lists)
two_opt_first(tour); % best improvement including speed-ups (dlbs, fixed radius near
      neighbour searches, neughbourhood lists)
three_opt_first(tour); % first improvement
```

Table 17.1 Cases for k-opt moves.

k	No. of Cases
2	1
3	4
4	20
5	148
6	1,358
7	15,104
8	198,144
9	2,998,656
10	51,290,496

[Appelgate Bixby, Chvátal, Cook, 2006]

Table 17.2 Computer-generated source code for k-opt moves.

k	No. of Lines
6	120,228
7	1,259,863
8	17,919,296

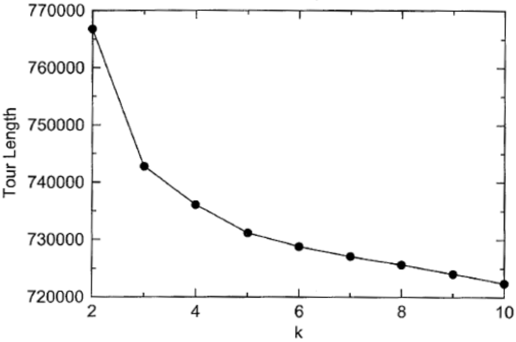


Figure 17.1 k-opt on a 10,000-city Euclidean TSP.

References

Applegate D.L., Bixby R.E., Chvátal V., and Cook W.J. (2006). **The Traveling Salesman Problem: A Computational Study**. Princeton University Press.

Bentley J. (1992). **Fast algorithms for geometric traveling salesman problems**. *ORSA Journal on Computing*, 4(4), pp. 387–411.

Johnson D.S. and McGeoch L.A. (2002). **Experimental analysis of heuristics for the STSP**. In *The Traveling Salesman Problem and Its Variations*, edited by G. Gutin and A. Punnen, pp. 369–443. Kluwer Academic Publishers, Boston, MA, USA.

Single Machine Total Weighted Tardiness Problem

- Interchange: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k

    $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k

    $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i-j|)$ evaluation each
  But possible to speed up with systematic examination by means of swaps: an interchange is equivalent to $|i-j|$ swaps hence overall examination takes $O(n^2)$