DM841

DISCRETE OPTIMIZATION

Part 2 – Heuristics

Efficiency Issues

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

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

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 $O(n^2)$

Outline

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

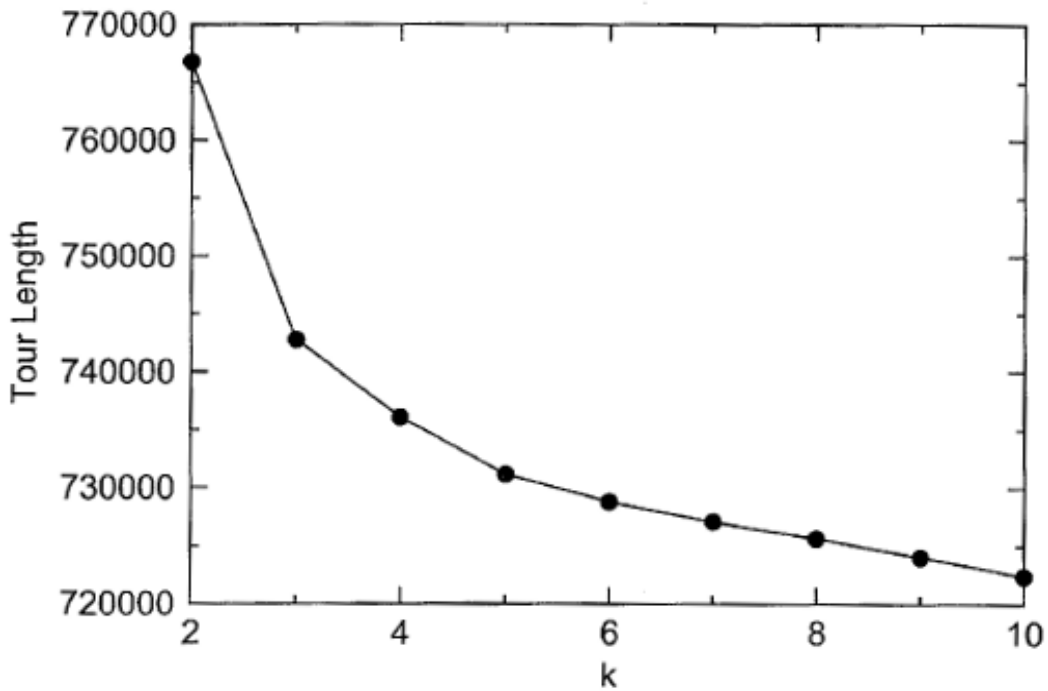


Figure 17.1 k-opt on a 10,000-city Euclidean TSP.

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 $w_i = 1$ 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

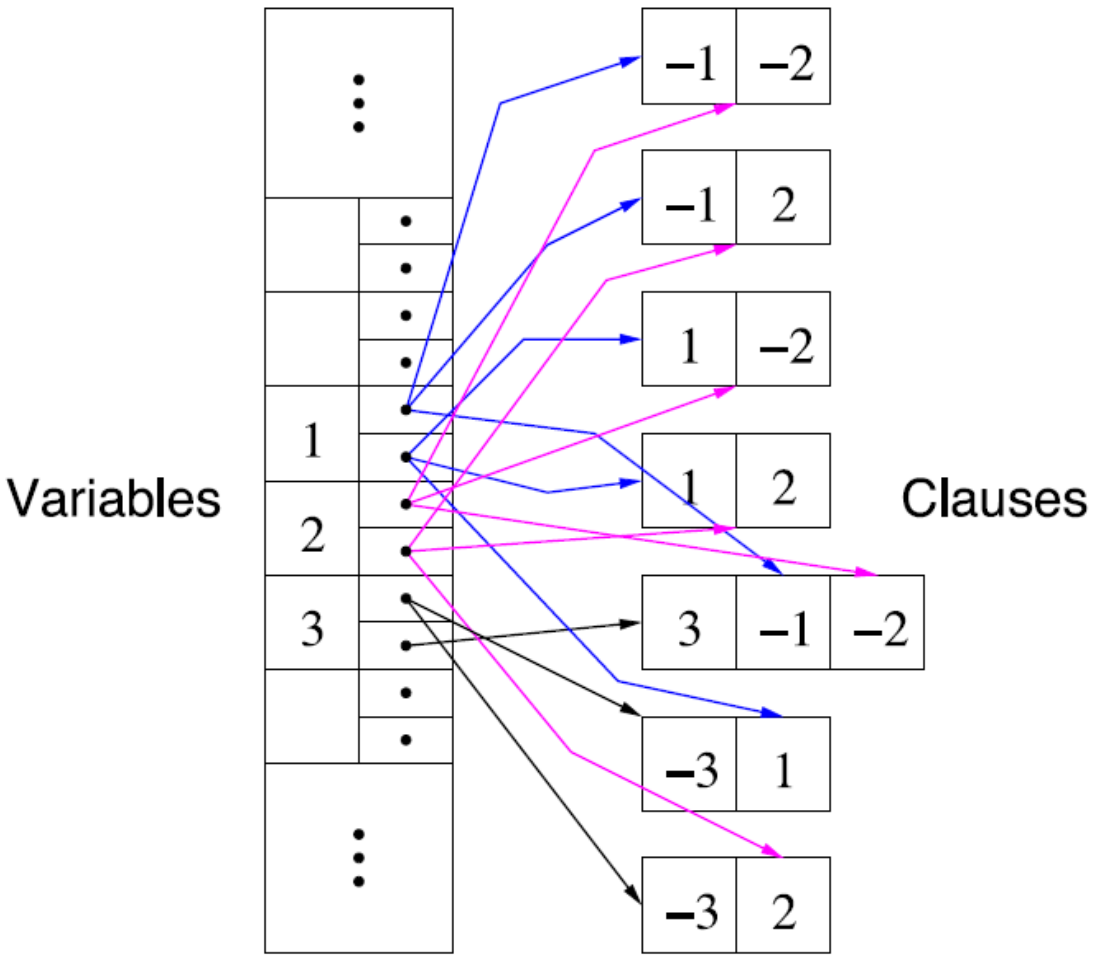
Examine Neighborhood

- choose the var with best score

Update:

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

$C(x_j)$ 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 **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 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 $x_l \in C_i$ **do**
    update score x_l depending on its critical status before flipping x_j

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.

Outline

Computational Complexity of LS

For a local search algorithm to be effective, search initialization and individual search steps should be efficiently computable.

Complexity class $\mathcal{PLS}$: class of problems for which a local search algorithm exists with polynomial time complexity for:

- search initialization
- any single search step, including computation of evaluation function value

For any problem in $\mathcal{PLS}$ ...

- local optimality can be verified in polynomial time
- improving search steps can be computed in polynomial time
- **but:** finding local optima may require super-polynomial time

Computational Complexity of LS

$\mathcal{PLS}$-complete: Among the most difficult problems in $\mathcal{PLS}$;
if for any of these problems local optima can be found
in polynomial time, the same would hold for all problems in $\mathcal{PLS}$.

Some complexity results:

- TSP with k-exchange neighborhood with $k > 3$
  is $\mathcal{PLS}$-complete.
- TSP with 2- or 3-exchange neighborhood is in $\mathcal{PLS}$, but
  $\mathcal{PLS}$-completeness is unknown.

Outline

Comments to Last Year Submissions

- Focus on relevant aspects, not on trivial and known features. For example, explaining the EasyLocal is not necessary as we all know about it. The algorithmic sketch must be on a relevant and original procedure. What you choose to describe and to show algorithmically will also be used to decide the grade.

- Be formal and do not use terms like "stupid". Do not tell about lack of time (everybody always lacks of time anyway).

- Do not make speculations but try to support your claims by experimental or analytic evidence.

- Define the notation that you use

- The calculation of the Delta by incremental updates is a requirement

- Recognize the algorithms that you end up implementing and give their name.

- Random restart is really a basic algorithm and you have to do better than that.

- You will get credit for how involved the method is and for the amount of work done.

- Check whether your algorithm has chances for ending in a loop. That would be bad.

- Your algorithms must be 100% reproducible by only reading the report.

- Write what you have done not what you would/could have done.

- Make a clear list of the algorithms you tested and give names to the algorithms you are describing. In this way it becomes clearer what you are precisely referring to. When writing the name of the algorithms use a different style, for example, sanserif or slanted, etc.

- Remember to give the big O analysis of the main procedures, that is, constructing a solution, initializing the data structures, evaluating a move, deciding a step in the local search and updating the data structures (ie, ensuring invariants).

- Write the report keeping in mind that the reader will be the external censor, Stefan Ropke. He is well acknowledged about local search and heuristics but has not been at the lectures and hence he does not know what we have precisely discussed about.

- Consider carefully the focus of your description and algorithmic sketches. This choice is alone providing to the examiners an indication of the level reached in this course. Reporting general sketches that have been seen in class is not a smart choice. A more appropriate choice is showing the specialized, non-trivial procedures that you have developed, that may indicate the originality and depth of thought in your work.

- Make it possible to distinguish entities in your plots to both readers that will print in colors and to those that will print in black and white.

- Leave a space before opening a parenthesis. Example: "Heuristics(DM841) wrong. "Heuristics (DM841)" is correct.