DM811 – Fall 2009
Heuristics for Combinatorial Optimization

Lecture 10
Efficient Local Search Exercises

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Resume

- Fromalization and properties of neighborhood operators
- Distances in neighborhood graph
- Non-obvious solution representations
  - parallel machine scheduling problem
  - Steiner tree problem

Knapsack, Bin Packing, Cutting Stock

Knapsack

Given: a knapsack with maximum weight W and a set of n items $\{1, 2, \ldots, n\}$, with each item j associated to a profit p_j and to a weight w_j.

Task: Find the subset of items of maximal total profit and whose total weight is not greater than W.

One dimensional Bin Packing

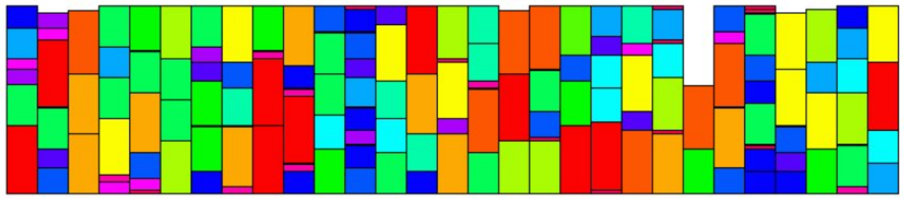
Given: A set $L = (a_1, a_2, \ldots, a_n)$ of *items*, each with a size $s(a_i) \in (0, 1]$ and an unlimited number of unit-capacity bins $B_1, B_2, \ldots, B_m$.

Task: Pack all the items into a minimum number of unit-capacity bins $B_1, B_2, \ldots, B_m$.

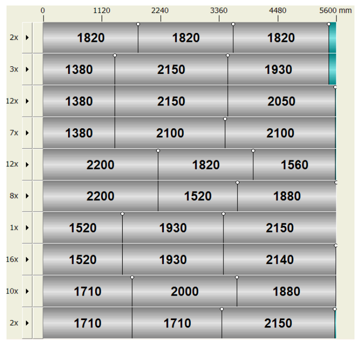
Cutting stock

Each item has a profit $p_j > 0$ and a number of times it must appear a_i. The task is to select a subset of items to be packed in a single finite bin that maximizes the total selected profit.

Bin Packing



Cutting Stock



Two-Dimensional Packing Problems

Two dimensional bin packing

Given: A set $L = (a_1, a_2, \ldots, a_n)$ of n rectangular *items*, each with a width w_j and a height h_j and an unlimited number of identical rectangular bins of width W and height H.
Task: Allocate all the items into a minimum number of bins, such that the original orientation is respected (no rotation of the items is allowed).

Two dimensional strip packing

Given: A set $L = (a_1, a_2, \ldots, a_n)$ of n rectangular *items*, each with a width w_j and a height h_j and a bin of width W and infinite height (*a strip*).
Task: Allocate all the items into the strip by minimizing the used height and such that the original orientation is respected (no rotation of the items is allowed).

Two dimensional cutting stock

Each item has a profit $p_j > 0$ and the task is to select a subset of items to be packed in a single finite bin that maximizes the total selected profit.

Three dimensional

Given: A set $L = (a_1, a_2, \ldots, a_n)$ of rectangular *boxes*, each with a width w_j, height h_j and depth d_j and an unlimited number of three-dimensional bins $B_1, B_2, \ldots, B_m$ of width W, height H, and depth D.

Task: Pack all the boxes into a minimum number of bins, such that the original orientation is respected (no rotation of the boxes is allowed)

Outline

1. Efficient Local Search
    Application Examples

Efficiency vs Effectiveness

The performance of local search is determined by:

1. quality of local optima (effectiveness)
2. time to reach local optima (efficiency):
   A. time to move from one solution to the next
   B. number of solutions to reach local optima

Note:

- Local minima depend on g and neighborhood function $\mathcal{N}$.
- Larger neighborhoods $\mathcal{N}$ induce
  - neighborhood graphs with smaller diameter;
  - fewer local minima.

Ideal case: exact neighborhood, *i.e.*, neighborhood function for which any local optimum is also guaranteed to be a global optimum.

- Typically, exact neighborhoods are too large to be searched effectively (exponential in size of problem instance).

Trade-off (to be assessed experimentally):

- Using larger neighborhoods can improve performance of II (and other LS methods).
- **But:** time required for determining improving search steps increases with neighborhood size.

Speedups Techniques for Efficient Neighborhood Search

1) Incremental updates
2) Neighborhood pruning

Speedups in Neighborhood Examination

1) Incremental updates (aka delta evaluations)

- **Key idea:** calculate effects of differences between current search position s and neighbors s' on evaluation function value.
- Evaluation function values often consist of independent contributions of solution components; hence, $f(s)$ can be efficiently calculated from $f(s')$ by differences between s and s' in terms of solution components.
- Typically crucial for the efficient implementation of II algorithms (and other LS techniques).

Do not do this:

```
tmp ← current
while ∃ unseen sol in N(current) do
    change current into sol
    evaluate current
    if current better than tmp then
        break;
    current ← tmp
```

Do this:

```
while ∃ unseen sol in N(current) do
    evaluate changes at current
    if improving then
        change current into sol
```

Example: Incremental updates for TSP

- solution components = edges of given graph G
- standard 2-exchange neighborhood, *i.e.*, neighboring round trips p, p' differ in two edges
- $w(p') := w(p)$ − edges in p but not in p'
  + edges in p' but not in p

Note: Constant time (4 arithmetic operations), compared to linear time (n arithmetic operations for graph with n vertices) for computing $w(p')$ from scratch.

2) Neighborhood Pruning

- **Idea:** Reduce size of neighborhoods by excluding neighbors that are likely (or guaranteed) not to yield improvements in f.
- **Note:** Crucial for large neighborhoods, but can be also very useful for small neighborhoods (*e.g.*, linear in instance size).

Example: Heuristic candidate lists for the TSP

- *Intuition:* High-quality solutions likely include short edges.
- Candidate list of vertex v: list of v's nearest neighbors (limited number), sorted according to increasing edge weights.
- Search steps (*e.g.*, 2-exchange moves) always involve edges to elements of candidate lists.
- Significant impact on performance of LS algorithms for the TSP.

Overview

Delta evaluations and neighborhood examinations in:

- Permutations
  - TSP
  - SMTWTP, Parallel Machine, Bin Packing
- Assignments
  - CSP, SAT, GCP, Bin Packing
- Sets
  - Set Covering, Max Independent Set, p-median

Example: Iterative Improvement for k-col

- **search space** S**:** set of all k-colorings of G
- **solution set** S'**:** set of all proper k-coloring of F
- **neighborhood function** $\mathcal{N}$**:** 1-exchange neighborhood (as in Uninformed Random Walk)
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $\texttt{init}\{\emptyset, \varphi'\} := 1/|S|$ for all colorings φ'
- **step function**:
  - **evaluation function:** $g(\varphi) :=$ number of edges in G whose ending vertices are assigned the same color under assignment φ (*Note:* $g(\varphi) = 0$ iff φ is a proper coloring of G.)
  - **move mechanism:** uniform random choice from improving neighbors, *i.e.*, $\texttt{step}\{\varphi, \varphi'\} := 1/|I(\varphi)|$ if $s' \in I(\varphi)$, and 0 otherwise, where $I(\varphi) := \{\varphi' \mid \mathcal{N}(\varphi, \varphi') \wedge g(\varphi') < g(\varphi)\}$
- **termination**: when no improving neighbor is available *i.e.*, $\texttt{terminate}\{\varphi, \top\} := 1$ if $I(\varphi) = \emptyset$, and 0 otherwise.

- 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

TSP data structures

Tour representation:

- `reverse(a, b)`
- `succ`
- `prec`
- `sequence(a,b,c)` – check whether b is within a and b

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/Lab/ls.c`

```
two_opt_b(tour);
two_opt_f(tour);
two_opt_best(tour);
two_opt_first(tour);
three_opt_first(tour);
```


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

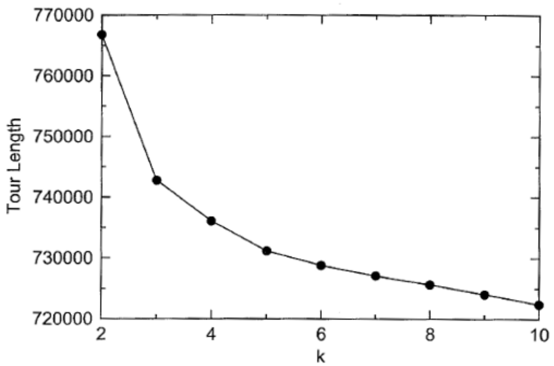


Figure 17.1 k-opt on a 10,000-city Euclidean TSP.

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 insert is equivalent to $|i-j|$ swaps hence overall examination takes $O(n^2)$

The Max Independent Set Problem

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 ($\omega : V \rightarrow \mathbf{R}$)
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 ($\omega : V \rightarrow \mathbf{R}$)
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

The p-median Problem

- **Given:**
  a set U of locations for n users
  a set F of locations of m facilities
  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.*,

$$\min_J \sum_{i \in U} \min_{j \in F} d_{ij} \qquad J \subseteq F \text{ and } |J| = p$$