DM811

Heuristics for Combinatorial Optimization

Lecture 16
Examples

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
- ✔ Local Search: Neighborhoods and Search Landscape
- ✔ Efficient Local Search: Incremental Updates and Neighborhood Pruning
- ✔ 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. Recap.

2. Other Combinatorial Optimization Problems
   - Quadratic Assignment Problem
   - School Scheduling
   - Linear Ordering
   - Bin Packing

Outline

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 $\texttt{terminate} : S_\pi \times M_\pi \to \{\top, \bot\}$

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

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

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

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

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 $O(n^2)$

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

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/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

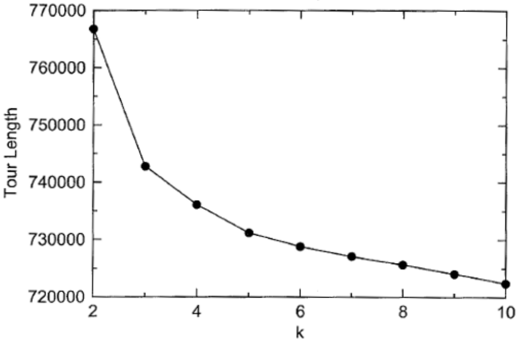


Figure 17.1 k-opt on a 10,000-city Euclidean TSP.

Outline

Outline

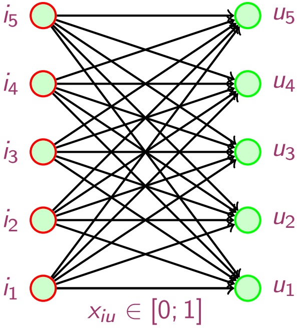
Quadratic Assignment Problem

- **Given:**
  n units with a matrix $F = [f_{ij}] \in \mathbf{R}^{n \times n}$ of flows between them and
  n locations with a matrix $D = [d_{uv}] \in \mathbf{R}^{n \times n}$ of distances
- **Task:** Find the assignment σ of units to locations that minimizes the sum of product between flows and distances, ie,

$$\min_{\sigma \in \Sigma} \sum_{i,j} f_{ij} d_{\sigma(i)\sigma(j)}$$

Applications: hospital layout; keyboard layout

Quadratic Programming Formulation



indices i, j for units and u, v for locations:

$$
\begin{array}{ll}
\min & \sum_i \sum_u \sum_j \sum_v f_{ij} d_{uv} x_{iu} x_{jv} + \left(\sum_i \sum_u c_{iu} x_{iu} \right) \\
\text{s.t.} & \sum_i x_{iu} = 1 \quad \forall u \\
& \sum_u x_{iu} = 1 \quad \forall i \\
& x \geq 0 \text{ and integer } \forall i, u
\end{array}
$$

Largest instances solvable exactly $n = 30$

Example: QAP

$$D = \begin{pmatrix} 0 & 4 & 3 & 2 & 1 \\ 4 & 0 & 3 & 2 & 1 \\ 3 & 3 & 0 & 2 & 1 \\ 2 & 2 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 2 & 3 & 4 \\ 2 & 2 & 0 & 3 & 4 \\ 3 & 3 & 3 & 0 & 4 \\ 4 & 4 & 4 & 4 & 0 \end{pmatrix}$$

The optimal solution is $\sigma = (1, 2, 3, 4, 5)$, that is,
facility 1 is assigned to location 1,
facility 2 is assigned to location 2, etc.

The value of $f(\sigma)$ is 100.

Delta evaluation

Evaluation of 2-exchange $\{r, s\}$ can be done in $O(n)$

$$\begin{aligned}\Delta(\psi, r, s) = b_{rr} \cdot (a_{\psi_s\psi_s} - a_{\psi_r\psi_r}) + b_{rs} \cdot (a_{\psi_s\psi_r} - a_{\psi_r\psi_s}) + \\ b_{sr} \cdot (a_{\psi_r\psi_s} - a_{\psi_s\psi_r}) + b_{ss} \cdot (a_{\psi_r\psi_r} - a_{\psi_s\psi_s}) + \\ \sum_{k=1, k\neq r,s}^{n} (b_{kr} \cdot (a_{\psi_k\psi_s} - a_{\psi_k\psi_r}) + b_{ks} \cdot (a_{\psi_k\psi_r} - a_{\psi_k\psi_s}) + \\ b_{rk} \cdot (a_{\psi_s\psi_k} - a_{\psi_r\psi_k}) + b_{sk} \cdot (a_{\psi_r\psi_k} - a_{\psi_s\psi_k}))\end{aligned}$$

Example: Tabu Search for QAP

- **Solution representation**: permutation π
- **Initial Solution**: randomly generated
- **Neighborhood**: interchange
  $\Delta_I : \quad \delta(\pi) = \{\pi' | \pi'_k = \pi_k \text{ for all } k \neq \{i,j\} \text{ and } \pi'_i = \pi_j, \pi'_j = \pi_i\}$
- **Tabu status**: forbid δ that place back the items in the positions they have already occupied in the last `tt` iterations (short term memory)

- Implementation details:
  - compute $f(\pi') - f(\pi)$ in $O(n)$ or $O(1)$ by storing the values all possible previous moves.
  - maintain a matrix $[T_{ij}]$ of size $n \times n$ and write the last time item i was moved in location k plus tt
  - δ is tabu if it satisfies both:
    - $T_{i,\pi(j)} \geq$ current iteration
    - $T_{j,\pi(i)} \geq$ current iteration

Example: Robust Tabu Search for QAP

- **Aspiration criteria**:
  - allow forbidden δ if it improves the last π^*
  - select δ if never chosen in the last A iterations (long term memory)
- Parameters: $\texttt{tt} \in [\lfloor 0.9n \rfloor, \lceil 1.1n + 4 \rceil]$ and $A = 5n^2$

Example: Reactive Tabu Search for QAP

- **Aspiration criteria**:
  - allow forbidden δ if it improves the last π^*
- **Tabu Tenure**
  - maintain a hash table (or function) to record previously visited solutions
  - increase `tt` by a factor $\alpha_{inc}(= 1.1)$ if the current solution was previously visited
  - decrease `tt` by a factor $\alpha_{dec}(= 0.9)$ if `tt` not changed in the last *sttc* iterations or all moves are tabu
- Trigger escape mechanism if a solution is visited more than $nr(= 3)$ times
- Escape mechanism $= 1 + (1 + r) \cdot ma/2$ random moves

Outline

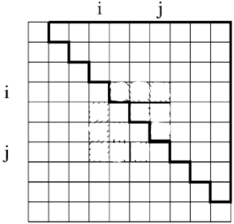
Linear Ordering Problem

Input: an $n \times n$ matrix C

Task: Find a permutation π of the column and row indices $\{1, \ldots, n\}$ such that the value

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\pi_i \pi_j}$$

is maximized. In other terms, find a permutation of the columns and rows of C such that the sum of the elements in the upper triangle is maximized.



Consider as an example the (5,5)-matrix:

$$H = \begin{bmatrix} 0 & 16 & 11 & 15 & 7 \\ 21 & 0 & 14 & 15 & 9 \\ 26 & 23 & 0 & 26 & 12 \\ 22 & 22 & 11 & 0 & 13 \\ 30 & 28 & 25 & 24 & 0 \end{bmatrix}$$

$\pi = (1, 2, 3, 4, 5)$. The sum of its superdiagonal elements is 138.
$\pi = (5, 3, 4, 2, 1)$ i.e., H_{12} becomes $H_{\pi(1)\pi(2)} = H_{54}$ in the permuted matrix.
Thus the optimal triangulation of H is

$$H^* = \begin{bmatrix} 0 & 25 & 24 & 28 & 30 \\ 12 & 0 & 26 & 23 & 26 \\ 13 & 11 & 0 & 22 & 22 \\ 9 & 14 & 15 & 0 & 21 \\ 7 & 11 & 15 & 16 & 0 \end{bmatrix}$$

Now the sum of superdiagonal elements is 247.

LOP Applications: Graph Theory

Definition: A directed graph (or digraph) D consists of a non-empty finite set $V(D)$ of distinct vertices and a finite set A of ordered pairs of distinct vertices called arcs.

LOP Applications: Graph Theory

Definition: A directed graph (or digraph) D consists of a non-empty finite set $V(D)$ of distinct vertices and a finite set A of ordered pairs of distinct vertices called arcs.

Feedback arc set problem (FASP)

Input: A directed graph $D = (V, A)$, where $V = \{1, 2, \ldots, n\}$, and arc weights c_{ij} for all $[ij] \in A$

Task: Find a permutation $\pi_1, \pi_2, \ldots \pi_n$ of V (that is, a linear ordering of V) such that the total costs of those arcs $[\pi_j \pi_i]$ where $j > i$ (that is, the arcs that point backwards in the ordering)

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\pi_j \pi_i}$$

is minimized.

LOP Applications: Graph Theory (2)

Definition: A linear ordering of a finite set of vertices $V = \{1, 2, \ldots, n\}$ is a bijective mapping (permutation) $\pi : \{1, 2, \ldots, n\} \rightarrow V$. For $u, v \in V$, we say that u is "before" v if $\pi^{-1}(u) < \pi^{-1}(v)$ ($\pi^{-1}(i) = pos_\pi(i)$).

LOP Applications: Graph Theory (2)

Definition: A linear ordering of a finite set of vertices $V = \{1, 2, \ldots, n\}$ is a bijective mapping (permutation) $\pi : \{1, 2, \ldots, n\} \to V$. For $u, v \in V$, we say that u is "before" v if $\pi^{-1}(u) < \pi^{-1}(v)$ ($\pi^{-1}(i) = pos_\pi(i)$).

Definition: A digraph D is complete if, for every pair x, y of distinct vertices of D both xy and yx arcs are in D.

LOP Applications: Graph Theory (2)

Definition: A linear ordering of a finite set of vertices $V = \{1, 2, \ldots, n\}$ is a bijective mapping (permutation) $\pi : \{1, 2, \ldots, n\} \to V$. For $u, v \in V$, we say that u is "before" v if $\pi^{-1}(u) < \pi^{-1}(v)$ ($\pi^{-1}(i) = pos_\pi(i)$).

Definition: A digraph D is complete if, for every pair x, y of distinct vertices of D both xy and yx arcs are in D.

Definition: An oriented graph is a digraph with no cycle of length two. A tournament is an oriented graph where every pair of distinct vertices are adjacent.

LOP Applications: Graph Theory (2)

Definition: A linear ordering of a finite set of vertices $V = \{1, 2, \ldots, n\}$ is a bijective mapping (permutation) $\pi : \{1, 2, \ldots, n\} \rightarrow V$. For $u, v \in V$, we say that u is "before" v if $\pi^{-1}(u) < \pi^{-1}(v)$ ($\pi^{-1}(i) = pos_{\pi}(i)$).

Definition: A digraph D is complete if, for every pair x, y of distinct vertices of D both xy and yx arcs are in D.

Definition: An oriented graph is a digraph with no cycle of length two. A tournament is an oriented graph where every pair of distinct vertices are adjacent.

Remark: Given a digraph $D = (V, A)$ and a linear ordering of the vertices V, the arc set $E = \{[uv] | \pi^{-1}(u) < \pi^{-1}(v)\}$ forms an acyclic tournament on V. Similarly, an acyclic tournament $T = (V, E)$ induces a linear ordering of V.

Definition: The cost of a linear ordering is expressed by

$$\sum_{\pi^{-1}(u)<\pi^{-1}(v)} c_{uv}$$

where the costs c_{uv} are the costs associated to the arcs.

Definition: The cost of a linear ordering is expressed by

$$\sum_{\pi^{-1}(u)<\pi^{-1}(v)} c_{uv}$$

where the costs c_{uv} are the costs associated to the arcs.

Linear Ordering Problem

Input: Given a complete digraph $D = (V, A)$ with arc weights c_{ij} for all $ij \in A$

Task: Find an acyclic tournament $T = (V, T)$ in D such that

$$f(T) = \sum_{ij \in T} c_{ij}$$

is maximized.

Aggregation of Individual Preferences

Kemeny's problem. Suppose that there are m persons and each person i, $i = 1, ..., m$, has ranked n objects by giving a linear ordering T_i of the objects. Which common linear ordering aggregates the individual orderings in the best possible way?

Aggregation of Individual Preferences

Kemeny's problem. Suppose that there are m persons and each person i, $i = 1, ..., m$, has ranked n objects by giving a linear ordering T_i of the objects. Which common linear ordering aggregates the individual orderings in the best possible way?

$\rightsquigarrow$ linear ordering problem by setting $c_{ij} =$ number of persons preferring object O_i to object O_j

LOP Applications: Economics

Input-output analysis (Leontief, Nobel prize)

The economy of a state is divided into n sectors, and an $n \times n$ input-output matrix C is constructed where the entry c_{ij} denotes the transactions from sector i to sector j in that year.

Triangulation (ie, solving associated LOP) allows identification of important inter-industry relations in an economy (from primary stage sectors via the manufacturing sectors to the sectors of final demand) and consequent comparisons between different countries.

Depicts dependencies between the different branches of an economy

Ranking in Sports Tournaments

H_{ij} = number of goals which were scored by team i against team j.

Table 1.1 Premier League 2006/2007 (left: official, right: triangulated)

1 Manchester United	1 Chelsea
2 Chelsea	2 Arsenal
3 Liverpool	3 Manchester United
4 Arsenal	4 Everton
5 Tottenham Hotspur	5 Portsmouth
6 Everton	6 Liverpool
7 Bolton Wanderers	7 Reading
8 Reading	8 Tottenham Hotspur
9 Portsmouth	9 Aston Villa
10 Blackburn Rovers	10 Blackburn Rovers
11 Aston Villa	11 Middlesborough
12 Middlesborough	12 Charlton Athletic
13 Newcastle United	13 Bolton Wanderers
14 Manchester City	14 Wigan Athletic
15 West Ham United	15 Manchester City
16 Fulham	16 Sheffield United
17 Wigan Athletic	17 Fulham
18 Sheffield United	18 Newcastle United
19 Charlton Athletic	19 Watford
20 Watford	20 West Ham United

R. Martí, G. Reinelt, R. Martí and G. Reinelt. *The Linear Ordering Problem*, *Introduction*. Springer Berlin Heidelberg, 2011, 1-15

Outline

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.

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 $B_1, B_2, \ldots, B_m$.

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

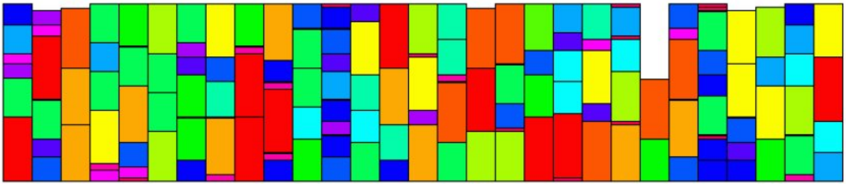
Given: A set $L = (a_1, a_2, \ldots, a_n)$ of *items*, each with a size $s(a_i) \in (0, 1]$ and an unlimited number of unit-capacity bins $B_1, B_2, \ldots, B_m$.

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

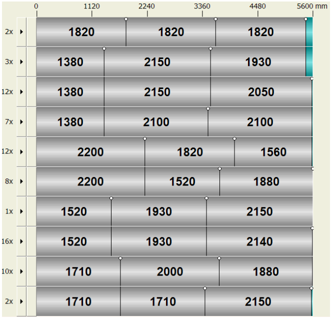
Cutting stock

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



Heuristics for Bin Packing

- Construction Heuristics
  - Best Fit Decreasing (BFD)
  - First Fit Decreasing (FFD) $C_{max}(FFD) \leq \frac{11}{9} C_{max}(OPT) + \frac{6}{9}$

Heuristics for Bin Packing

- Construction Heuristics
  - Best Fit Decreasing (BFD)
  - First Fit Decreasing (FFD) $C_{max}(FFD) \leq \frac{11}{9} C_{max}(OPT) + \frac{6}{9}$
- Local Search: [Alvim and Aloise and Glover and Ribeiro, 1999]

  Step 1: remove one bin and redistribute items by BFD

  Step 2: if infeasible, re-make feasible by redistributing items for pairs of bins, such that their total weights becomes equal (number partitioning problem)

[Levine and Ducatelle, 2004]

The solution before local search (the bin capacity is 10):

The bins:	| 3 3 3 | 6 2 1 | 5 2 | 4 3 | 7 2 | 5 4 |

Open the two smallest bins:

Remaining:	| 3 3 3 | 6 2 1 | 7 2 | 5 4 |
Free items:	5, 4, 3, 2

Try to replace 2 current items by 2 free items, 2 current by 1 free or 1 current by 1 free:

First bin:	3 3 3 $\rightarrow$ 3 5 2	new free: 4, 3, 3, 3
Second bin:	6 2 1 $\rightarrow$ 6 4	new free: 3, 3, 3, 2, 1
Third bin:	7 2 $\rightarrow$ 7 3	new free: 3, 3, 2, 2, 1
Fourth bin:	5 4 stays the same	

Reinsert the free items using FFD:

Fourth bin:	5 4 $\rightarrow$ 5 4 1
Make new bin:	3 3 2 2
Final solution:	| 3 5 2 | 6 4 | 7 3 | 5 4 1 | 3 3 2 2 |

Repeat the procedure: no further improvement possible

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 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 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)

List of Problems

See http://www.nada.kth.se/~viggo/problemlist/

Outline

School scheduling

Input: a finite set of time periods and courses with assigned: a teacher, a set of attending students and a suitable room.

Task: Produce weekly timetable of courses, that is: assign a time period of the week (typically one hour) to every course such that courses are assigned to different time periods if:

- they are taught by the same teacher
- they can be held only in the same room
- they share students.