DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS FOR COMBINATORIAL OPTIMZATION

Lecture 6

Local Search

Marco Chiarandini

slides based on
http://www.sls-book.net/
H. Hoos and T. Stützle, 2005

Outline

The Single Machine Total Tardiness Problem

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 $T_i = \{C_i - d_i, 0\}$ (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 $\pi = J_3, J_1, J_5, J_4, J_1, J_6$

Job	J_3	J_1	J_5	J_4	J_1	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

Outline

1. Local Search
   - Introduction
   - Components
   - Iterative Improvement
   - Neighborhoods Representations

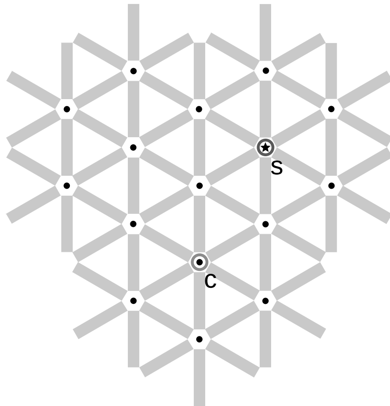
Local Search Paradigm

- search space = complete candidate solutions
- search step = modification of one or more solution components
- iteratively generate and evaluate candidate solutions
  - decision problems: evaluation = test if solution
  - optimization problems: evaluation = check objective function value
- evaluating candidate solutions is typically computationally much cheaper than finding (optimal) solutions

Iterative Improvement (II):

determine initial candidate solution s
while s has better neighbors **do**
  choose a neighbor s' of s such that $f(s') < f(s)$
  $s := s'$

Local search — global view



- vertices: candidate solutions (search positions)
- vertex labels: evaluation function
- edges: connect "neighboring" positions
- s: (optimal) solution
- c: current search position

Definitions: Local Search Algorithm (1)

Given a (combinatorial) optimization problem Π and one of its instances π:

- search space $S(\pi)$
  specified by candidate solution representation:
  discrete structures: sequences, permutations, graphs, partitions
  (*e.g.*, for SAT: array (sequence of all truth assignments to propositional variables)

  Note: solution set $S'(\pi) \subseteq S(\pi)$
  (*e.g.*, for SAT: models of given formula)
- evaluation function $f(\pi) : S(\pi) \mapsto \mathbb{R}$
  (*e.g.*, for SAT: number of false clauses)
- neighborhood function, $\mathcal{N}(\pi) : S \mapsto 2^{S(\pi)}$
  (*e.g.*, for SAT: neighboring variable assignments differ in the truth value of exactly one variable)

Definition: Local Search Algorithm (2)

- set of memory states $M(\pi)$
  (may consist of a single state, for LS algorithms that do not use memory)
- initialization function $\texttt{init} : \emptyset \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
  (specifies probability distribution over initial search positions and memory states)
- step function $\texttt{step} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
  (maps each search position and memory state onto probability distribution over subsequent, neighboring search positions and memory states)
- termination predicate $\texttt{terminate} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(\{\top, \bot\})$
  (determines the termination probability for each search position and memory state)

procedure *LS-Decision*(π)
input: *problem instance* $\pi \in \Pi$
output: *solution* $s \in S'(\pi)$ **or** $\emptyset$
$(s, m) := \texttt{init}(\pi)$;

while not $\texttt{terminate}(\pi, s, m)$ **do**
$(s, m) := \texttt{step}(\pi, s, m)$;
end

if $s \in S'(\pi)$ **then**
return s
else
return $\emptyset$
end
end *LS-Decision*

procedure *LS-Minimization*(π')
input: *problem instance* $\pi' \in \Pi'$
output: *solution* $s \in S'(\pi')$ **or** $\emptyset$
$(s, m) := \texttt{init}(\pi')$;
$\hat{s} := s$;
while not $\texttt{terminate}(\pi', s, m)$ **do**
$(s, m) := \texttt{step}(\pi', s, m)$;
if $f(\pi', s) < f(\pi', \hat{s})$ **then**
$\hat{s} := s$;
end
end
if $\hat{s} \in S'(\pi')$ **then**
return $\hat{s}$
else
return $\emptyset$
end
end *LS-Minimization*

Definition: Local Search Algorithm

For given problem instance π:

- search space $S(\pi)$
- solution set $S'(\pi) \subseteq S(\pi)$
- neighborhood relation $\mathcal{N}(\pi) \subseteq S(\pi) \times S(\pi)$
- evaluation function $f(\pi) : S \mapsto \mathbf{R}$
- set of memory states $M(\pi)$
- initialization function $\texttt{init} : \emptyset \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
- step function $\texttt{step} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
- termination predicate $\texttt{terminate} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(\{\top, \bot\})$

Example: Uninformed random walk for SAT

- **search space** S**:** set of all truth assignments to variables in given formula F (**solution set** S'**:** set of all models of F)
- **neighborhood relation** $\mathcal{N}$**:** *1-flip neighborhood*, *i.e.*, assignments are neighbors under $\mathcal{N}$ iff they differ in the truth value of exactly one variable
- **evaluation function** not used, or $f(s) = 0$ if model $f(s) = 1$ otherwise
- **memory:** not used, *i.e.*, $M := \{0\}$

Example: Uninformed random walk for SAT (continued)

- **initialization:** uniform random choice from S, *i.e.*, $\mathtt{init}(,\{a', m\}) := 1/|S|$ for all assignments a' and memory states m
- **step function:** uniform random choice from current neighborhood, *i.e.*, $\mathtt{step}(\{a, m\}, \{a', m\}) := 1/|N(a)|$ for all assignments a and memory states m, where $N(a) := \{a' \in S \mid \mathcal{N}(a, a')\}$ is the set of all neighbors of a.
- **termination:** when model is found, *i.e.*, $\mathtt{terminate}(\{a, m\}, \{\top\}) := 1$ if a is a model of F, and 0 otherwise.

Definition: LS Algorithm Components (continued)

Search Space

Defined by the solution representation:

- permutations
  - linear (scheduling)
  - circular (TSP)
- arrays (assignment problems: GCP)
- sets or lists (partition problems: Knapsack)

Definition: LS Algorithm Components (continued)

Neighborhood function

Also defined as: $\mathcal{N} : S \times S \to \{T, F\}$ or $\mathcal{N} \subseteq S \times S$

- neighborhood (set) of candidate solution s: $N(s) := \{s' \in S \mid \mathcal{N}(s, s')\}$
- neighborhood size is $|N(s)|$
- neighborhood is symmetric if: $s' \in N(s) \Rightarrow s \in N(s')$
- neighborhood graph of (S, f, N, π) is a directed vertex-weighted graph: $G_{\mathcal{N}}(\pi) := (V, A)$ with $V = S(\pi)$ and $(uv) \in A \Leftrightarrow v \in N(u)$ (if symmetric neighborhood $\Rightarrow$ undirected graph)
- Solution j is reachable from solution i if neighborhood graph has a path from i to j.
- strongly connected neighborhood graph
- weakly optimally connected neighborhood graph

A neighborhood function is also defined by means of an operator.

An operator Δ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(s) \iff \exists\, \delta \in \Delta, \delta(s) = s'$$

Definition

k-exchange neighborhood: candidate solutions s, s' are neighbors iff s differs from s' in at most k solution components

Examples:

- 1-exchange (flip) neighborhood for SAT (solution components = single variable assignments)
- 2-exchange neighborhood for TSP (solution components = edges in given graph)

Definition: LS Algorithm Components (continued)

Note:

- Local search implements a walk through the neighborhood graph
- Procedural versions of `init`, `step` and `terminate` implement sampling from respective probability distributions.
- Memory state m can consist of multiple independent attributes, *i.e.*, $M(\pi) := M_1 \times M_2 \times \ldots \times M_{l(\pi)}$.
- Local search algorithms are *Markov processes*: behavior in any search state $\{s, m\}$ depends only on current position s and (limited) memory m.

Definition: LS Algorithm Components (continued)

Search step (or move):
pair of search positions s, s' for which
s' can be reached from s in one step, *i.e.*, $\mathcal{N}(s, s')$ and
$\mathtt{step}(\{s, m\}, \{s', m'\}) > 0$ for some memory states $m, m' \in M$.

- Search trajectory: finite sequence of search positions $< s_0, s_1, \ldots, s_k >$ such that (s_{i-1}, s_i) is a *search step* for any $i \in \{1, \ldots, k\}$ and the probability of initializing the search at s_0 is greater zero, *i.e.*, $\mathtt{init}(\{s_0, m\}) > 0$ for some memory state $m \in M$.
- Search strategy: specified by `init` and `step` function; to some extent independent of problem instance and other components of LS algorithm.
  - random
  - based on evaluation function
  - based on memory

Uninformed Random Picking

- $\mathcal{N} := S \times S$
- does not use memory and evaluation function
- `init`, `step`: uniform random choice from S, *i.e.*, for all $s, s' \in S$, $\mathtt{init}(s) := \mathtt{step}(\{s\}, \{s'\}) := 1/|S|$

Uninformed Random Walk

- does not use memory and evaluation function
- `init`: uniform random choice from S
- `step`: uniform random choice from current neighborhood, *i.e.*, for all $s, s' \in S$, $\mathtt{step}(\{s\}, \{s'\}) := 1/|N(s)|$ if $\mathcal{N}(s, s')$, and 0 otherwise

Note: These uninformed LS strategies are quite ineffective, but play a role in combination with more directed search strategies.

Definition: LS Algorithm Components (continued)

Evaluation (or cost) function:

- function $f(\pi) : S(\pi) \mapsto \mathbb{R}$ that maps candidate solutions of a given problem instance π onto real numbers, such that global optima correspond to solutions of π;
- used for ranking or assessing neighbors of current search position to provide guidance to search process.

Evaluation *vs* objective functions:

- *Evaluation function*: part of LS algorithm.
- *Objective function*: integral part of optimization problem.
- Some LS methods use evaluation functions different from given objective function (*e.g.*, dynamic local search).

Iterative Improvement

- does not use memory
- `init`: uniform random choice from S
- `step`: uniform random choice from improving neighbors, *i.e.*, $\mathtt{step}(\{s\},\{s'\}) := 1/|I(s)|$ if $s' \in I(s)$, and 0 otherwise, where $I(s) := \{s' \in S \mid \mathcal{N}(s,s') \text{and} f(s') < f(s)\}$
- terminates when no improving neighbor available (to be revisited later)
- different variants through modifications of step function (to be revisited later)

Note: II is also known as *iterative descent* or *hill-climbing*.

Example: Iterative Improvement for SAT

- **search space** S**:** set of all truth assignments to variables in given formula F
  (**solution set** S'**:** set of all models of F)
- **neighborhood relation** $\mathcal{N}$**:** 1-flip neighborhood (as in Uninformed Random Walk for SAT)
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $\mathtt{init}(\emptyset, \{a'\}) := 1/|S|$ for all assignments a'
- **evaluation function:** $f(a) :=$ number of clauses in F that are *unsatisfied* under assignment a
  (*Note:* $f(a) = 0$ iff a is a model of F.)
- **step function**: uniform random choice from improving neighbors, *i.e.*, $\mathtt{step}(a, a') := 1/\#I(a)$ if $s' \in I(a)$, and 0 otherwise, where $I(a) := \{a' \mid \mathcal{N}(a,a') \wedge f(a') < f(a)\}$
- **termination**: when no improving neighbor is available *i.e.*, $\mathtt{terminate}(a, \top) := 1$ if $I(a) = \emptyset$, and 0 otherwise.

Definition:

- Local minimum: search position without improving neighbors w.r.t. given evaluation function f and neighborhood $\mathcal{N}$, *i.e.*, position $s \in S$ such that $f(s) \leq f(s')$ for all $s' \in N(s)$.
- Strict local minimum: search position $s \in S$ such that $f(s) < f(s')$ for all $s' \in N(s)$.
- *Local maxima* and *strict local maxima*: defined analogously.

There might be more than one neighbor that have better cost.

Pivoting rule decides which to choose:

- Best Improvement (aka *gradient descent*, *steepest descent*, *greedy hill-climbing*): Choose maximally improving neighbor, *i.e.*, randomly select from $I^*(s) := \{s' \in N(s) \mid f(s') = g^*\}$, where $g^* := \min\{f(s') \mid s' \in N(s)\}$.

  Note: Requires evaluation of all neighbors in each step.
- First Improvement: Evaluate neighbors in fixed order, choose first improving step encountered.

  Note: Can be much more efficient than Best Improvement; order of evaluation can have significant impact on performance.

Iterative Improvement (2 OPT)

```
procedure TSP-2opt-first(s)
    input: an initial candidate tour s ∈ S(∈)
    output: a local optimum s ∈ S(π)
    Δ = 0;
    do
        Improvement=FALSE;
        for i = 1 to n − 2 do
        if i = 1 then n′ = n − 1 elsen′ = n
            for j = i + 2 to n′ do
                Δij = d(ci, cj) + d(ci+1, cj+1) − d(ci, ci+1) − d(cj, cj+1)
                if Δij < 0 then
                    UpdateTour(s,i,j);
                    Improvement=TRUE;
            end
        end
    until Improvement==TRUE;
end TSP-2opt-first
```


Example: Random-order first improvement for the TSP

- **Given:** TSP instance G with vertices $v_1, v_2, \ldots, v_n$.
- search space: Hamiltonian cycles in G;
  use standard 2-exchange neighborhood
- **Initialization:**
  search position := fixed canonical path $< v_1, v_2, \ldots, v_n, v_1 >$
  P := random permutation of $\{1, 2, \ldots, n\}$
- **Search steps:** determined using first improvement w.r.t. $f(p)$ = weight of path p, evaluating neighbors in order of P (does not change throughout search)
- **Termination:** when no improving search step possible (local minimum)

Example: Random order first improvement for SAT

```
procedure URW-for-SAT(F,maxSteps)
    input: propositional formula F, integer maxSteps
    output: model of F or ∅
    choose assignment φ of truth values to all variables in F
        uniformly at random;
    steps := 0;
    while not((φ satisfies F) and (steps < maxSteps)) do
        select x uniformly at random from {x′|x′ is a variable in F and
        changing value of x′ in φ decreases the number of unsatisfied clauses};
        steps := steps+1;
    end
    if φ satisfies F then
        return φ
    else
        return ∅
    end
end URW-for-SAT
```


Solution Representations and Neighborhoods

Three different types of solution representations:

- *Permutation*
  - *linear permutation*: Single Machine Total Weighted Tardiness Problem
  - *circular permutation*: Traveling Salesman Problem
- *Assignment*: Graph Coloring Problem, SAT, CSP
- *Set, Partition*: Max Independent Set

A neighborhood function $\mathcal{N} : S \to S \times S$ is also defined through an operator. An operator Δ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(s) \quad \Longleftrightarrow \quad \exists \delta \in \Delta | \delta(s) = s'$$

Permutations

$\Pi(n)$ indicates the set all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2 \ldots, n)$ is the identity permutation ι.

If $\pi \in \Pi(n)$ and $1 \leq i \leq n$ then:

- π_i is the element at position i
- $pos_\pi(i)$ is the position of element i

Alternatively, a permutation is a bijective function $\pi(i) = \pi_i$

the permutation product $\pi \cdot \pi'$ is the composition $(\pi \cdot \pi')_i = \pi'(\pi(i))$

For each π there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$

$$\Delta_N \subset \Pi$$

Neighborhood Operators for Linear Permutations

Swap operator

$$\Delta_S = \{\delta_S^i | 1 \leq i \leq n\}$$

$$\delta_S^i(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i+1} \pi_i \ldots \pi_n)$$

Interchange operator

$$\Delta_X = \{\delta_X^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_X^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{i+1} \ldots \pi_{j-1} \pi_i \pi_{j+1} \ldots \pi_n)$$

Insert operator

$$\Delta_I = \{\delta_I^{ij} | 1 \leq i \leq n, 1 \leq j \leq n, j \neq i\}$$

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_n) & i > j \end{cases}$$

Neighborhood Operators for Circular Permutations

Reversal (2-edge-exchange)

$$\Delta_R = \{\delta_R^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_R^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \ldots \pi_i \pi_{j+1} \ldots \pi_n)$$

Block moves (3-edge-exchange)

$$\Delta_B = \{\delta_B^{ijk} | 1 \leq i < j < k \leq n\}$$

$$\delta_B^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \ldots \pi_k \pi_i \ldots \pi_{j-1} \pi_{k+1} \ldots \pi_n)$$

Short block move (Or-edge-exchange)

$$\Delta_{SB} = \{\delta_{SB}^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_{SB}^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_i \ldots \pi_{j-1} \pi_{j+3} \ldots \pi_n)$$

Neighborhood Operators for Assignments

An assignment can be represented as a mapping
$\sigma : \{X_1 \ldots X_n\} \rightarrow \{v : v \in D, |D| = k\}$:

$$\sigma = \{X_i = v_i, X_j = v_j, \ldots\}$$

One-exchange operator

$$\Delta_{1E} = \{\delta_{1E}^{il} | 1 \leq i \leq n, 1 \leq l \leq k\}$$

$$\delta_{1E}^{il}(\sigma) = \left\{\sigma : \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j) \ \forall j \neq i\right\}$$

Two-exchange operator

$$\Delta_{2E} = \{\delta_{2E}^{ij} | 1 \leq i < j \leq n\}$$

$$\delta_{2E}^{ij}\left\{\sigma : \sigma'(X_i) = \sigma(X_j),\ \sigma'(X_j) = \sigma(X_i) \text{ and } \sigma'(X_l) = \sigma(X_l) \ \forall l \neq i, j\right\}$$

Neighborhood Operators for Partitions or Sets

An assignment can be represented as a partition of objects selected and not selected $s : \{X\} \to \{C, \overline{C}\}$
(it can also be represented by a bit string)

One-addition operator

$$\Delta_{1E} = \{\delta^{v}_{1E} | v \in \overline{C}\}$$

$$\delta^{v}_{1E}(s) = \{s : C' = C \cup v \text{ and } \overline{C}' = \overline{C} \setminus v\}$$

One-deletion operator

$$\Delta_{1E} = \{\delta^{v}_{1E} | v \in C\}$$

$$\delta^{v}_{1E}(s) = \{s : C' = C \setminus v \text{ and } \overline{C}' = \overline{C} \cup v\}$$

Swap operator

$$\Delta_{1E} = \{\delta^{v}_{1E} | v \in C, u \in \overline{C}\}$$

$$\delta^{v}_{1E}(s) = \{s : C' = C \cup u \setminus v \text{ and } \overline{C}' = \overline{C} \cup v \setminus u\}$$