DM865 – Spring 2018

Heuristics and Approximation Algorithms

(Stochastic) Local Search Algorithms

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Definitions

2. Local Search Algorithms

3. Local Search Revisited
    - Components

Outline

Definitions
Neighborhood function

Neighborhood function $N : S_\pi \to 2^S$

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, N, π) is a directed graph: $G_N := (V, A)$ with $V = S$ and $(uv) \in A \Leftrightarrow v \in N(u)$
  (if symmetric neighborhood $\leadsto$ undirected graph)

A neighborhood function is also defined by means of an operator (aka move).

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

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

Definition

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

Examples:

- 2-exchange neighborhood for TSP
  (solution components = edges in given graph)

Definitions

Definition:

- Local minimum: search position without improving neighbors wrt given evaluation function f and neighborhood function 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.

Outline

1. Definitions
2. Local Search Algorithms
3. Local Search Revisited
   - Components

Local Search

- Model
  - Variables $\rightsquigarrow$ solution representation, search space
  - Constraints:
    – implicit
    – one-way defining invariants
    – soft
  - evaluation function
- Search (solve an optimization problem)
  - Construction heuristics
  - Neighborhoods, Iterative Improvement, (Stochastic) local search
  - Metaheuristics: Tabu Search, Simulated Annealing, Iterated Local Search
  - Population based metaheuristics

Local Search Algorithms

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

1. search space $S(\pi)$
   - specified by the definition of (finite domain, integer) variables and their values handling implicit constraints
   - all together they determine the representation of candidate solutions
   - common solution representations are discrete structures such as: sequences, permutations, partitions, graphs

   Note: solution set $S'(\pi) \subseteq S(\pi)$

Local Search Algorithms (cntd)

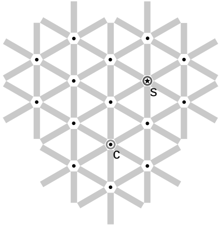
2. evaluation function $f_\pi : S(\pi) \rightarrow \mathbf{R}$
   - it handles the soft constraints and the objective function
3. neighborhood function, $N_\pi : S \rightarrow 2^{S(\pi)}$
   - defines for each solution $s \in S(\pi)$ a set of solutions $N(s) \subseteq S(\pi)$ that are in some sense close to s.

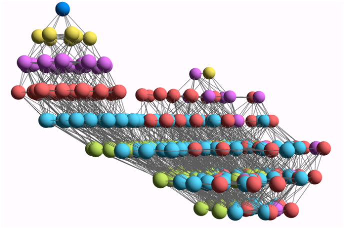
Local Search Algorithms (cntd)

Further components [according to [HS]]

4. set of memory states $M(\pi)$
   (may consist of a single state, for LS algorithms that do not use memory)
5. initialization function $\texttt{init} : \emptyset \to S(\pi)$
   (can be seen as a probability distribution $\Pr(S(\pi) \times M(\pi))$ over initial search positions and memory states)
6. step function $\texttt{step} : S(\pi) \times M(\pi) \to S(\pi) \times M(\pi)$
   (can be seen as a probability distribution $\Pr(S(\pi) \times M(\pi))$ over subsequent, neighboring search positions and memory states)
7. termination predicate $\texttt{terminate} : S(\pi) \times M(\pi) \to \{\top, \bot\}$
   (determines the termination state for each search position and memory state)

Local search — global view





Neighborhood graph

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

Local Search Algorithms

Note:

- Local search implements a walk through the neighborhood graph
- Procedural versions of `init`, `step` and `terminate` implement sampling from respective probability distributions.
- Local search algorithms can be described as Markov processes:
  behavior in any search state $\{s, m\}$ depends only
  on current position s
  higher order MP if (limited) memory m.

Local Search (LS) Algorithm Components

Step function

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

- Search trajectory: finite sequence of search positions $\langle s_0, s_1, \ldots, s_k \rangle$ 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 than zero, *i.e.*, $\texttt{init}(\{s_0, m\}) > 0$
  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

Iterative Improvement

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

- If more than one neighbor has better cost then need to choose one (heuristic pivot rule)
- The procedure ends in a local optimum $\hat{s}$:
  Def.: Local optimum $\hat{s}$ w.r.t. N if $f(\hat{s}) \leq f(s)\ \forall s \in N(\hat{s})$
- Issue: how to avoid getting trapped in bad local optima?
  - use more complex neighborhood functions
  - restart
  - allow non-improving moves

Metaheuristics

- "Restart" + parallel search
  Avoid local optima
  Improve search space coverage

- Variable Neighborhood Search and Large Scale Neighborhood Search
  diversified neighborhoods + incremental algorithmics
  ("diversified" $\equiv$ multiple, variable-size, and rich).

- Tabu Search: Online learning of moves
  Discard undoing moves,
  Discard inefficient moves
  Improve efficient moves selection

- Simulated annealing
  Allow degrading solutions

Summary: Local Search Algorithms

For given problem instance π:

1. search space S_π, solution representation: variables + implicit constraints
2. evaluation function $f_\pi : S \to \mathbf{R}$, soft constraints + objective
3. neighborhood relation $\mathcal{N}_\pi \subseteq S_\pi \times S_\pi$
4. set of memory states M_π
5. initialization function $\mathtt{init} : \emptyset \to S_\pi \times M_\pi$
6. step function $\mathtt{step} : S_\pi \times M_\pi \to S_\pi \times M_\pi$
7. termination predicate $\mathtt{terminate} : S_\pi \times M_\pi \to \{\top, \bot\}$

Decision vs Minimization

LS-Decision(π)
input: problem instance $\pi \in \Pi$
output: solution $s \in S'(\pi)$ **or** $\emptyset$

$(s, m) := \texttt{init}(\pi)$

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

if $s \in S'(\pi)$ **then**
  return s
else
  return $\emptyset$

LS-Minimization(π')
input: problem instance $\pi' \in \Pi'$
output: solution $s \in S'(\pi')$ **or** $\emptyset$

$(s, m) := \texttt{init}(\pi')$;
$s_b := s$;
while not $\texttt{terminate}(\pi', \texttt{s}, \texttt{m})$ **do**
  $(s, m) := \texttt{step}(\pi', \texttt{s}, \texttt{m})$;
  if $f(\pi', s) < f(\pi', s_b)$ **then**
    $s_b := s$;

if $s_b \in S'(\pi')$ **then**
  return s_b
else
  return $\emptyset$

However, the algorithm on the left has little guidance, hence most often decision problems are transformed in optimization problems by, eg, couting number of violations.

Outline

LS Algorithm Components

Search space

Search Space

Solution representations defined by the variables and the implicit constraints:

- permutations (implicit: alldiffrerent)
  - linear (scheduling problems)
  - circular (traveling salesman problem)
- arrays (implicit: assign exactly one, assignment problems: GCP)
- sets (implicit: disjoint sets, partition problems: graph partitioning, max indep. set)

$\leadsto$ Multiple viewpoints are useful in local search!

LS Algorithm Components

Evaluation function

Evaluation (or cost) function:

- function $f_\pi : S_\pi \to \mathbf{Q}$ that maps candidate solutions of a given problem instance π onto rational numbers (most often integer), such that global optima correspond to solutions of π;
- used for assessing or ranking 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.*, guided local search).

Constrained Optimization Problems

Constrained Optimization Problems exhibit two issues:

- feasibility
  eg, treveling salesman problem with time windows: customers must be visited within their time window.
- optimization
  minimize the total tour.

How to combine them in local search?

- sequence of feasibility problems
- staying in the space of feasible candidate solutions
- considering feasible and infeasible configurations

Constraint-based local search

From Van Hentenryck and Michel

If infeasible solutions are allowed, we count violations of constraints.

What is a violation?
Constraint specific:

- decomposition-based violations
  number of violated constraints, eg: alldiff
- variable-based violations
  min number of variables that must be changed to satisfy c.
- value-based violations
  for constraints on number of occurences of values
- arithmetic violations
- combinations of these

Constraint-based local search

From Van Hentenryck and Michel

Combinatorial constraints

- $\texttt{alldiff}(x_1, \ldots, x_n)$:
  Let a be an assignment with values $V = \{a(x_1), \ldots, a(x_n)\}$ and $c_v = \#_a(v, x)$ be the number of occurrences of v in a.
  Possible definitions for violations are:
  - $\text{viol} = \sum_{v \in V} I(\max\{c_v - 1, 0\} > 0)$ value-based
  - $\text{viol} = \max_{v \in V} \max\{c_v - 1, 0\}$ value-based
  - $\text{viol} = \sum_{v \in V} \max\{c_v - 1, 0\}$ value-based
  - $\#$ variables with same value, variable-based, here leads to same definitions as previous three

Arithmetic constraints

- $l \leq r \rightsquigarrow \text{viol} = \max\{l - r, 0\}$
- $l = r \rightsquigarrow \text{viol} = |l - r|$
- $l \neq r \rightsquigarrow \text{viol} = 1$ if $l = r$, 0 otherwise