DM811
Heuristics for Combinatorial Optimization

Lecture 7
Local Search: Further Analysis

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

LS Algorithm Components

Search Space

Defined by the solution representation:

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

LS Algorithm Components

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)

Notation: N when set, $\mathcal{N}$ when collection of sets or function

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)

LS Algorithm Components

Definition:

- Local minimum: search position without improving neighbors wrt 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.

LS Algorithm Components

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.

LS Algorithm Components

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

LS Algorithm Components

Evaluation (or cost) function:

- function $f(\pi) : S(\pi) \mapsto \mathbf{R}$ 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.*, guided local search).

Iterative Improvement

- does not use memory
- `init`: uniform random choice from S or construction heuristic
- `step`: uniform random choice from improving neighbors

$$\Pr(s, s') = \begin{cases} 1/|I(s)| \text{ if } s' \in I(s) \\ 0 \text{ otherwise} \end{cases}$$

where $I(s) := \{s' \in S \mid \mathcal{N}(s, s') \text{ and } f(s') < f(s)\}$
$I(s)$ can not be maximal (see next slide)

- terminates when no improving neighbor available

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

Iterative Improvement (cntd)

Pivoting rule decides which neighbors go in $I(s)$

- Best Improvement (aka *gradient descent*, *steepest descent*, *greedy hill-climbing*): Choose maximally improving neighbors,
  i.e., $I(s) := \{s' \in N(s) \mid f(s') = g^*\}$,
  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 one encountered.

  Note: Can be more efficient than Best Improvement but not in the worst case; order of evaluation can impact performance.

Examples

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
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $\texttt{init}(\emptyset, \{a\}) := 1/|S|$ 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.*, $\texttt{step}(a, a') := 1/|I(a)|$ if $a' \in I(a)$, 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.*, $\texttt{terminate}(a, \top) := 1$ if $I(a) = \emptyset$, and 0 otherwise.

Examples

Random order first improvement for SAT

URW-for-SAT(F,*maxSteps*)
input: *propositional formula* F, *integer maxSteps*
output: *a model for* F **or** $\emptyset$

choose assignment φ of truth values to all variables in F
  uniformly at random;
steps := 0;
while $\neg(\varphi$ 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*$+1$;
if φ satisfies F **then**
  return φ
else
  return $\emptyset$

Examples

Iterative Improvement for TSP

TSP-2opt-first(s)
input: *an initial candidate tour* $s \in S(\in)$
output: *a local optimum* $s \in S(\pi)$
$\Delta = 0$;
for $i = 1$ to $n-2$ **do**
  if $i = 1$ **then** $n' = n-1$ **else** $n' = n$
  for $j = i+2$ to n' **do**
    $\Delta_{ij} = d(c_i, c_j) + d(c_{i+1}, c_{j+1}) - d(c_i, c_{i+1}) - d(c_j, c_{j+1})$
    if $\Delta_{ij} < 0$ **then**
      $\mathtt{UpdateTour(s,i,j)}$

is it really?

Examples

Iterative Improvement for TSP

TSP-2opt-first(s)
input: *an initial candidate tour* $s \in S(\in)$
output: *a local optimum* $s \in S(\pi)$
$\Delta = 0$;
Improvement=*TRUE*;
while Improvement==TRUE **do**
  Improvement=*FALSE*;
  for $i = 1$ to $n-2$ **do**
    if $i = 1$ **then** $n' = n-1$ **else** $n' = n$
    for $j = i+2$ to n' **do**
      $\Delta_{ij} = d(c_i, c_j) + d(c_{i+1}, c_{j+1}) - d(c_i, c_{i+1}) - d(c_j, c_{j+1})$
      if $\Delta_{ij} < 0$ **then**
        $\mathtt{UpdateTour(s,i,j)}$
        Improvement=*TRUE*

Examples

Random-order first improvement for the TSP

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

Examples

Graph Coloring and Constraint Satisfaction

Different choices for the candidate solutions, neighborhood structures and evaluation function define different approaches to the problem

k-fixed	complete	proper	
k-fixed	partial	proper	$+$
k-fixed	complete	unproper	$+++$
k-fixed	partial	unproper	$-$
k-variable	complete	proper	$++$
k-variable	partial	proper	$-$
k-variable	complete	unproper	$++$
k-variable	partial	unproper	$-$