DM841

Discrete Optimization

Part 2 – Lecture 4
Beyond Local Search

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Resumé: Constraint-Based Local Search

Constraint-Based Local Search = Modelling + Search

Resumé: Local Search Modelling

Optimization problem (decision problems $\mapsto$ optimization):

- Parameters
- Variables and Solution Representation
  implicit constraints
- Soft constraint violations
- Evaluation function: soft constraints + objective function

Differentiable objects:

- Neighborhoods
- Delta evaluations
  Invariants defined by one-way constraints

Resumé: Local Search Algorithms

A theoretical framework

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

Computational analysis on each of these components is necessay!

Resumé: Local Search Algorithms

- Random Walk
- First/Random Improvement
- Best Improvement
- Min Conflict Heuristic

The step is the component that changes. It is also called: pivoting rule (for allusion to the simplex for LP)

Examples: TSP

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;
- **Neighborhood relation** N**:** standard 2-exchange neighborhood
- **Initialization:**

  search position := fixed canonical tour $< v_1, v_2, \ldots, v_n, v_1 >$
  "mask" P := random permutation of $\{1, 2, \ldots, n\}$

- **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: TSP

Iterative Improvement for TSP

TSP-2opt-first(s)
input: *an initial candidate tour* $s \in S(\in)$
output: *a local optimum* $s \in S_\pi$

for $i = 1$ to $n - 1$ **do**
  for $j = i + 1$ to n **do**
    if $P[i] + 1 \geq n$ or $P[j] + 1 \geq n$ **then** *continue* ;
    if $P[i] + 1 = P[j]$ or $P[j] + 1 = P[i]$ **then** *continue* ;

$$\begin{aligned}\Delta_{ij} \quad = \quad & d(\pi_{P[i]}, \pi_{P[j]}) + d(\pi_{P[i]+1}, \pi_{P[j]+1}) + \\ & -d(\pi_{P[i]}, \pi_{P[i]+1}) - d(\pi_{P[j]}, \pi_{P[j]+1})\end{aligned}$$

    if $\Delta_{ij} < 0$ **then**
      $\texttt{UpdateTour}(\texttt{s}, \texttt{P[i]}, \texttt{P[j]})$

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* $s \in S_\pi$

```
FoundImprovement:=TRUE;
while FoundImprovement do
    FoundImprovement:=FALSE;
    for i = 1 to n − 1 do
        for j = i + 1 to n do
            if P[i] + 1 ≥ n or P[j] + 1 ≥ n then continue ;
            if P[i] + 1 = P[j] or P[j] + 1 = P[i] then continue ;
            Δij = d(π_P[i], π_P[j]) + d(π_P[i]+1, π_P[j]+1)+
                  −d(π_P[i], π_P[i]+1) − d(π_P[j], π_P[j]+1)
            if Δij < 0 then
                UpdateTour(s, P[i], P[j])
                FoundImprovement=TRUE
```

$$\Delta_{ij} = d(\pi_{P[i]}, \pi_{P[j]}) + d(\pi_{P[i]+1}, \pi_{P[j]+1}) + -d(\pi_{P[i]}, \pi_{P[i]+1}) - d(\pi_{P[j]}, \pi_{P[j]+1})$$

Outline

1. Local Search Revisited
   Components

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)

$\rightsquigarrow$ Multiple viewpoints are useful also 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:
  - $\mathrm{viol} = \sum_{v \in V} I(\max\{c_v - 1, 0\} > 0)$ value-based
  - $\mathrm{viol} = \max_{v \in V} \max\{c_v - 1, 0\}$ value-based
  - $\mathrm{viol} = \sum_{v \in V} \max\{c_v - 1, 0\}$ value-based
  - $\#$ variables with same value, variable-based, here leads to same definitions as previous three

Arithmetic constraints

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