DM841

Discrete Optimization — Heuristics

Large Neighorhood Search

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

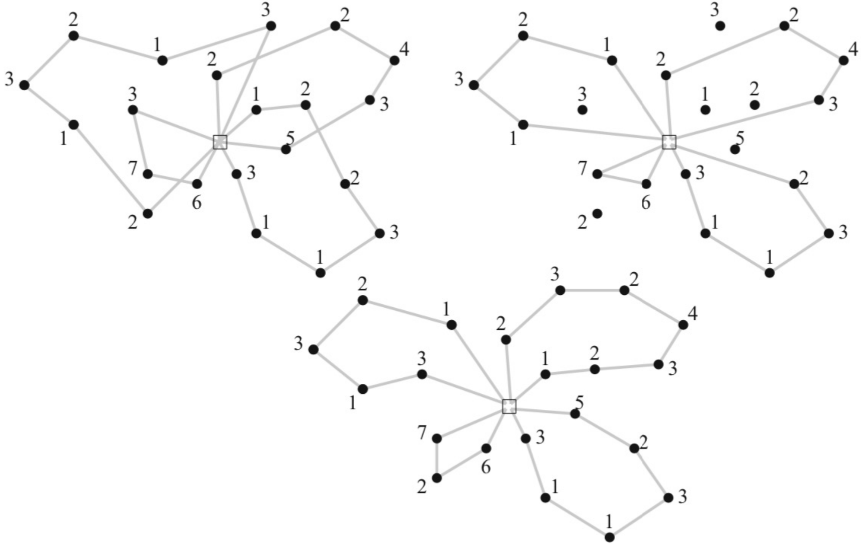
1. Adaptive Large Neighborhood Search (ALNS)
2. Construct, Merge, Solve & Adapt (CMSA)

Heuristic VLSN Algorithms

Other heuristics that can be seen as belonging to the class of VLSN are those metaheuristics based on alternately destroying and repairing the solution:

- Iterated greedy
- Large Neighborhood Search (LNS) proposed by [Shaw, 1998]
- Adaptive Large Neighborhood Search (ALNS) [Røpke and Pisinger, 2006]

Large Neighborhood Search



Large Neighborhood Search

input: a feasible solution x
$x^b = x$;
repeat
    $x^t = r(d(x))$;
    if accept(x^t, x) **then**
        $x = x^t$;
    end if
    if $c(x^t) < c(x^b)$ **then**
        $x^b = x^t$;
    end if
until stopping criterion is met
return x^b

The LNS metaheuristic does not search the entire neighborhood of a solution, but merely samples this neighborhood.

Acceptance criterion:

- Always
- Only if better
- record-to-record travel (accept if $f(s') \leq (1 + r)f(s_b)$)
- threshold accepting (Metropolis criterion)
- simulated annealing criterion

Degree of Destruction

- gradually increase
- randomly chosen from a specific range dependent on the instance size

To guarantee connectivity, it must be possible to destroy every part of the solution.

Repair method:

- problem-specific heuristic
- exact method
- general purpose mixed integer programming (MIP) (aka, fix and optimize)
- constraint programming solver (aka, fix and optimize)

It should allow diversification

Adaptive Large Neighborhood Search (ALNS)

Key Idea: allow multiple destroy and repair methods controlling with an adaptive weighting system how often a particular method is attempted during the search. [Ropke, Pisinger, 2006]

input: a feasible solution x
$x^b = x; \rho^- = (1, \ldots, 1); \rho^+ = (1, \ldots, 1);$
repeat
    select destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using ρ^- and ρ^+;
    $x^t = r(d(x));$
    if accept(x^t, x) **then**
        $x = x^t;$
    end if
    if $c(x^t) < c(x^b)$ **then**
        $x^b = x^t;$
    end if
    update ρ^- and ρ^+;
until stopping criterion is met
return x^b

Selection mechanism: roulette wheel principle:

$$p(j) = \frac{\rho_j^-}{\sum\limits_{k \in \Omega^-} \rho_k^-}$$

Update mechanism

$$\Psi = \max \begin{cases} \omega_1 & \text{if the new solution is a new global best} \\ \omega_2 & \text{if the new solution is better than the current one} \\ \omega_3 & \text{if the new solution is accepted} \\ \omega_4 & \text{if the new solution is rejected} \end{cases}$$

with normally $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4 \geq 0$. Only accepted a and b are updated:

$$\rho_a^- = \lambda \rho_a^- + (1 - \lambda)\Psi, \qquad \rho_b^+ = \lambda \rho_b^+ + (1 - \lambda)\Psi$$

$\lambda \in [0, 1]$ is a decay parameter

Design Choices

Destroy methods:

- Diversification: random destroy method.
- Intensification: remove q “critical” variables, i.e. variables having a large cost or variables that spoil the current structure of the solution (e.g. edges crossing each other in an Euclidean TSP). This is known as worst destroy or critical destroy.
- related destroy select a set of customers that have a high mutual relatedness measure. Eg on the CVRP, relatedness measure between each pair of customers is distance between the customers (and it could include customer demand)
- history based destroy q variables are chosen according to some historical information,

Repair methods:

- Greedy heuristics, problem specific
- include local search
- exact algorithms
- Mixed integer programming (aka, matheuristic)
- constraint programming

Design Choices

Large multiple-neighborhood search (LMNS) heuristics: It may be sufficient to have a number of destroy and repair heuristics that are selected randomly with equal probability, that is, without the adaptive layer.

Same robustness as ALNS heuristics, while fewer parameters to calibrate.

Other Relations

- Variable Neighborhood Search
- Portfolio Algorithms
- Hyperheuristics, another thread in UK
- Reinforcement learning

Outline

1. Adaptive Large Neighborhood Search (ALNS)

2. Construct, Merge, Solve & Adapt (CMSA)

Construct, Merge, Solve & Adapt (CMSA)

- $\mathcal{I}$ problem instance to a generic problem $\mathcal{P}$,
- C set of all possible components of which solutions to the problem instance are composed (eg, each combination of a variable with one of its values is a solution component)
- S valid solution to $\mathcal{I}$ is represented as a subset of the solution components C, that is, $S \subseteq C$.
- $C' \subseteq C$ contains the solution components that belong to a restricted problem instance, that is, a sub-instance (aka a domain tightening) of $\mathcal{I}$

Example, the input graph in case of the TSP. The set of all edges can be regarded as the set of all possible solution components C. The edges belonging to a tour S – that is, a valid solution – form the set of solution components that are contained in S. The union of the edges belonging to many different tours constitues the set C'.

Construct, Merge, Solve & Adapt (CMSA)

input: problem instance $\mathcal{I}$, values for parameters n_a and $\text{age}_{\max}$
S_{bsf}:=NULL, C':=$\emptyset$
$\text{age}[c]$:=0 for all $c \in C$
while CPU time limit not reached **do**
  for $i = 1, \ldots, n_a$ **do**
    S:=ProbabilisticSolutionGeneration(C)
    for all $c \in S$ **and** $c \notin C'$ **do**
      $\text{age}[c]$:=0
      C':=$C' \cup \{c\}$
    end for
  end for
  S'_{opt}:=ApplyExactSolver(C')
  if S'_{opt} is better than S_{bsf} **then** S_{bsf}:=S'_{opt}
  Adapt(C', S'_{opt}, $\text{age}_{\max}$)
end while
output: S_{bsf}

[Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization C Blum, P Pinacho, M López-Ibáñez, JA Lozano, COR, 2016]

Example: minimum common string partition (MCSP)

Given:

- Two input strings s_1 and s_2 of length n over a finite alphabet Σ.
- Two strings are related: each letter appears the same number of times in each of them. (hence $|s_1| = |s_2| = n$.)
- A valid solution is a partitioning s_1 into a set P_1 of non-overlapping substrings, and s_2 into a set P_2 of non-overlapping substrings, such that $P_1 = P_2$.

Goal:

- Find a valid solution such that $|P_1| = |P_2|$ is minimal.

Example

$s_1 = AGACTG$ and $s_2 = ACTAGG$.
The two strings are related.
A trivial valid solution is $P_1 = P_2 = \{A, A, C, T, G, G\}$. The objective function value of this solution is 6.
The optimal solution is $P_1 = P_2 = \{ACT, AG, G\}$ with objective function value 3.

- $C = \{c_1, \dots c_m\}$ be the arbitrarily ordered set of all possible common blocks of s^1 and s^2, i.e., C is the set of all solution components.

- a common block c_i of input strings s^1 and s^2 is denoted as a triple $\langle t_i; k_i^1; k_i^2 \rangle$, t_i is a string starting at position $1 \leq k_i^1 \leq n$ in string s^1 and starting at position $1 \leq k_i^2 \leq n$ in string s^2.

- a subset S of C is called a valid subset iff the following conditions hold
  1. $\sum_{c_i \in S} |t_i| \leq n$, that is, the sum of the length of the strings corresponding to the common blocks in S is smaller or equal to the length of the input strings.
  2. For any two common blocks $c_i, c_j \in S$ it holds that their corresponding strings neither overlap in s^1 nor in s^2.

- Given a valid subset $S \subset C$, set $Ext(S) \subset C \setminus S$ denotes the set of common blocks that may be used in order to extend S such that the result is again a valid subset.

ProbabilisticSolutionGeneration(C)

input: s^1, s^2, d_{rate}, l_{size}
$S := \emptyset$
while $Ext(S) \neq \emptyset$ **do**
  Choose a random number $\delta \in [0, 1]$
  if $\delta \leq d_{\text{rate}}$ **then**
    Choose c^* such that $|t_{c^*}| \geq |t_c|$ for all $c \in Ext(S)$
    $S{:=}S \cup \{c^*\}$
  else
    Let $L \subseteq Ext(S)$ contain the (at most) l_{size} longest common blocks from $Ext(S)$
    Choose c^* uniformly at random from L
    $S{:=}S \cup \{c^*\}$
  end if
end while
output: The complete (valid) solution S

ApplyExactSolver(C’): Solving reduced sub-instances

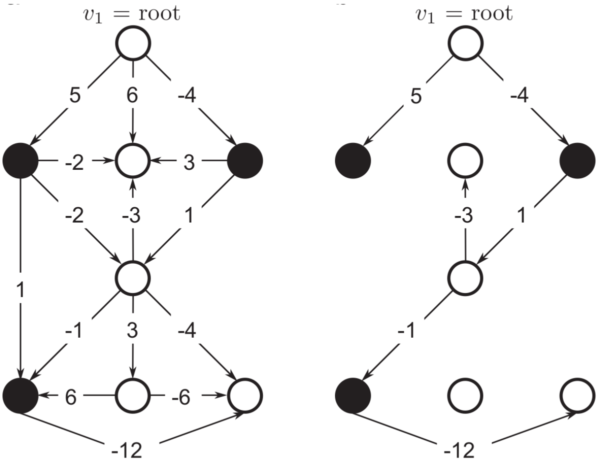
MIPped.

Example: minimum covering arborescence (MCA)

Given: directed (acyclic) graph $D = (V, A)$ with integer weights on the arcs $w(a) \in \mathbb{Z}$.

Task: Find subgraphs of minimal total weight that are arborescences rooted in a pre-defined root node v_1

arborescence: directed, rooted (not necessarily spanning) tree in which all arcs point away from the root node



ProbabilisticSolutionGeneration(C)

- complete set of solution components corresponds to the set A of arcs of the input graph, that is, $C = A$
- valid subset $S \subset A$ iff $T = (V(S), S)$ is an arborescence of the input graph G rooted in v_1
- $Ext(S) \subset A \setminus S$ arcs that can be added

input: a DAG $G = (V, A)$ with root node v_1, $d_{\min}$, $d_{\max}$
$S := \emptyset$
$\hat{A} := \mathrm{Out}(v_1)$
while $\hat{A} \neq \emptyset$ **do**
  $a^* := \mathrm{Choose}(\hat{A}, d_{\min}, d_{\max})$
  $S := S \cup \{a^*\}$
  $\hat{A} := Ext(S)$
  $\hat{A} := \mathrm{Reduce}(\hat{A})$
end while
output: valid subset S which induces arborescence $T = (V(S), S)$

ApplyExactSolver(C'): Solving reduced sub-instances

MIPped.