Heuristic Algorithms

Master's Degree in Computer Science/Mathematics

Roberto Cordone

DI - Università degli Studi di Milano

Schedule:	Thursday 14.30 - 16.30 on MS-Teams
	Friday 14.30 - 16.30 on MS-Teams
Office hours:	on appointment
E-mail:	roberto.cordone@unimi.it
Web page:	https://homes.di.unimi.it/cordone/courses/2020-ae/2020-ae.html
Ariel site:	https://rcordoneha.ariel.ctu.unimi.it

Lesson 18: *VND* and *DLS* Milano, A.A. 2020/21

Extending the local search without worsening

Instead of repeating the local search, extend it beyond the local optimum

To avoid worsening solutions, the selection step must be modified

$$\tilde{x} := \arg \min_{x' \in N(x)} f(x')$$

and two main strategies allow to do that

- the *Variable Neighbourhood Descent* (*VND*) changes the neighbourhood N
  - it guarantees an evolution with no cycles (*the objective improves*)
  - it terminates when all neighbourhoods have been exploited
- the *Dynamic Local Search* (*DLS*) changes the objective function f ($\tilde{x}$ *is better than* x *for the new objective, possibly worse for the old*)
  - it can be trapped in loops (*the new objective changes over time*)
  - it can proceed indefinitely

Variable Neighbourhood Descent (VND)

The *Variable Neighbourhood Descent* of Hansen and Mladenović (1997) exploits the fact that a solution is locally optimal for a specific neighbourhood

- a local optimum can be improved using a different neighbourhood

Given a family of neighbourhoods $N_1, \dots, N_{k_{\max}}$

1. set $k := 1$
2. apply a *steepest descent* exchange heuristic and find a local optimum $\bar{x}$ with respect to N_k
3. flag all neighbourhoods for which $\bar{x}$ is locally optimal and update k
4. if $\bar{x}$ is a local optimum for all N_k, terminate; otherwise, go back to point 2

Algorithm VariableNeighbourhoodDescent$(I, x^{(0)})$

$\mathrm{flag}_k := \text{false } \forall k$;

$\bar{x} := x^{(0)}$; $x^* := x^{(0)}$; $k := 1$;

While $\exists k : \mathrm{flag}_k = \text{false}$ *do*

  $\bar{x} :=$ SteepestDescent$(\bar{x}, k)$;

  If $f(\bar{x}) < f(x^*)$

    then $x^* := \bar{x}$; $\mathrm{flag}_{k'} := \text{false } \forall k' \neq k$;

    else $\mathrm{flag}_k := \text{true}$;

  $k :=$ *Update*(k);

EndWhile;

Return $(x^*, f(x^*))$;

VND and *VNS*

There is of course a strict relation between *VND* and *VNS*

(*in fact, they were proposed in the same paper*)

The fundamental differences are that in the *VND*

- at each step the current solution is the best known one
- the neighbourhoods are explored,
  instead of being used to extract random solutions

  They are never huge
- the neighbourhoods do not necessarily form a hierarchy

  The update of k is not always an increment
- when a local optimum for each N_k has been reached, terminate

  VND is deterministic and would not find anything else

Neighbourhood update strategies for the *VND*

There are two main classes of *VND* methods

- methods with heterogeneous neighbourhoods
  - exploit the potential of topologically different neighbourhoods (e.g., exchange vertices instead of edges)

  Consequently, k periodically scans the values from 1 to $k_{\max}$ (*possibly randomly permuting the sequence at each repetition*)
- methods with hierarchical neighbourhoods ($N_1 \subset \ldots \subset N_{k_{\max}}$)
  - fully exploit the small and fast neighbourhoods
  - resort to the large and slow ones only to get out of local optima (*usually terminating SteepestDescent prematurely*)

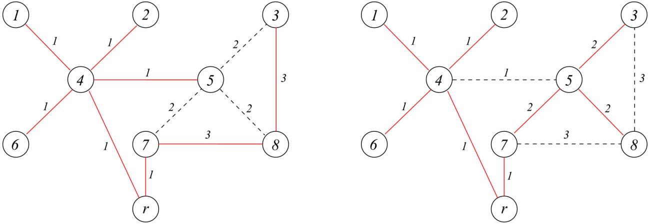
  Consequently, the update of k works as in the *VNS*
  - when no improvements can be found in N_k, increase k
  - when improvements can be found in N_k, decrease k back to 1

Terminate when the current solution is a local optimum for all N_k

- in the heterogeneous case, terminate when all fail
- in the hierarchical case, terminate when the largest fails

Example: the CMSTP

This instance of *CMSTP* has $n = 9$ vertices, uniform weights ($w_v = 1$), capacity $W = 5$ and the reported costs (the missing edges have $c_e \gg 3$)



Consider neighbourhood N_{S_1} (single-edge swaps) for the first solution:

- no edge in the right branch can be deleted
  because the left branch has zero residual capacity
  and a direct connection to the root would increase the cost
- deleting any edge in the left branch increases the total cost:
  the solution is a local optimum for N_{S_1}

Neighbourhood N_{T_1} (single-vertex transfers) has an improving solution, obtained moving vertex 5 from the left branch to the right one

Dynamic Local Search (DLS)

The *Dynamic Local Search* is also known as *Guided Local Search*

Its approach is complementary to *VND*

- it keeps the starting neighbourhood
- it modifies the objective function

It is often used when the objective is useless because it has wide *plateaus*

The basic idea is to

- define a penalty function $w : X \rightarrow \mathbb{N}$
- build an auxiliary function $\tilde{f}(f(x), w(x))$
  which combines the objective function f with the penalty w
- apply a *steepest descent* exchange heuristic to optimise $\tilde{f}$
- at each iteration update the penalty w based on the results

The penalty is adaptive in order to move away from recent local optima
but this introduces the risk of cycling

General scheme of the *DLS*

Algorithm DynamicLocalSearch$(I, x^{(0)})$

$w :=$ StartingPenalty(I);

$\bar{x} := x^{(0)}$; $x^* := x^{(0)}$;

While Stop() $=$ *false do*

  $(\bar{x}, x_f) :=$ SteepestDescent$(\bar{x}, f, w)$;

  If $f(x_f) < f(x^*)$ *then* $x^* := x_f$;

  $w :=$ *UpdatePenalty*$(w, \bar{x}, x^*)$;

EndWhile;

Return $(x^*, f(x^*))$;

Notice that the *steepest descent* heuristic

- optimises a combination $\tilde{f}$ of f and w
- returns two solutions:
  1. a final solution $\bar{x}$, locally optimal with respect to $\tilde{f}$, to update w
  2. a solution x_f, that is the best known with respect to f

Variants

The penalty can be applied (for example)

- additively to the elements of the solution:

$$\tilde{f}(x) = f(x) + \sum_{i \in x} w_i$$

- multiplicatively to components of the objective $f(x) = \sum_j \phi_j(x)$:

$$\tilde{f}(x) = \sum_j w_j \, \phi_j(x)$$

The penalty can be updated

- at each single neighbourhood exploration
- when a local optimum for $\tilde{f}$ is reached
- when the best known solution x^* is unchanged for a long time

The penalty can be modified with

- random updates: "noisy" perturbation of the costs
- memory-based updates, favouring the most frequent elements (intensification) or the less frequent ones (diversification)

Example: *DLS* for the *MCP*

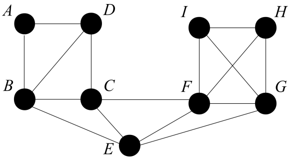
Given a undirected graph, find a maximum cardinality clique

- the exchange heuristic is a *VND* using the neighbourhoods
  1. N_{A_1} (vertex addition): the solution always improves, but the neighbourhood is very small and often empty
  2. N_{S_1} (exchange of an internal vertex with an external one): the neighbourhood is larger, but forms a *plateau* (uniform objective)
- the objective provides no useful direction in either neighbourhood
- associate to each vertex i a penalty w_i initially equal to zero
- the exchange heuristic minimises the total penalty (within the neighbourhood!)
- update the penalty
  1. when the exploration of N_{S_1} terminates: the penalty of the current clique vertices increases by 1
  2. after a given number of explorations: all the nonzero penalties decrease by 1

The rationale of the method consists in aiming to

- expel the internal vertices (diversification)
- in particular, the oldest internal vertices (memory)

Example: *DLS* for the *MCP*



Start from $x^{(0)} = \{B, C, D\}$, with $w = [\,0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\,]$

1. $w(\{B, C, E\}) = w(\{A, B, D\}) = 2$, but $\{A, B, D\}$ wins lexicographically: $x^{(1)} = \{A, B, D\}$ with $w = [\,1\ 2\ 1\ 2\ 0\ 0\ 0\ 0\ 0\,]$
2. $x^{(2)} = \{B, C, D\}$ with $w = [\,1\ 3\ 2\ 3\ 0\ 0\ 0\ 0\ 0\,]$ is the only neighbour
3. $w(\{B, C, E\}) = 5 < 7 = w(\{A, B, D\})$: $x^{(3)} = \{B, C, E\}$ with $w = [\,1\ 4\ 3\ 3\ 1\ 0\ 0\ 0\ 0\,]$
4. $w(\{C, E, F\}) = 4 < 10 = w(\{B, C, D\})$: $x^{(4)} = \{C, E, F\}$ with $w = [\,1\ 4\ 4\ 3\ 2\ 1\ 0\ 0\ 0\,]$
5. $w(\{E, F, G\}) = 3 < 11 = w(\{B, C, E\})$: $x^{(5)} = \{E, F, G\}$ with $w = [\,1\ 4\ 4\ 3\ 3\ 2\ 1\ 0\ 0\,]$
6. $w(\{F, G, H\}) = w(\{F, G, I\}) = 3 < 9 = w(\{C, E, F\})$: $x^{(6)} = \{F, G, H\}$ with $w = [\,1\ 4\ 4\ 3\ 3\ 3\ 2\ 1\ 0\,]$

Now the neighbourhood N_{A_1} is not empty: $x^{(7)} = \{F, G, H, I\}$

Example: *DLS* for the *MAX-SAT*

Given m logical disjunctions depending on n logical variables, find a truth assignment satisfying the maximum number of formulae

- neighbourhood N_{F_1} (1-flip) is generated complementing a variable
- associate to each logical formula a penalty w_j initially equal to 1
  (*each component is a satisfied formula*)
- the exchange heuristic maximizes the weight of satisfied formulae thus modifying their number with the multiplicative penalty
- the penalty is updated
  1. increasing the weight of unsatisfied formulae to favour them

     $$w_j := \alpha_{\text{us}}\ w_j \text{ for each } j \in U(x) \quad (\text{with } \alpha_{\text{us}} > 1)$$

     when a local optimum is reached
  2. reducing the penalty towards 1

     $$w_j := (1-\rho)\ w_j + \rho \cdot 1 \text{ for each } j \in C \quad (\text{with } \rho \in (0,1))$$

     with a certain probability or after a certain number of updates

Example: *DLS* for the *MAX-SAT*

The rationale of the method consists in aiming to

- satisfy the currently unsatisfied formulae (diversification)
- in particular, those which have been unsatisfied for longer time and more recently (memory)

The parameters tune intensification and diversification

- small values of α_{us} and ρ preserve the current penalty (intensification)
- large values of α_{us} and ρ cancel the current penalty (diversification)