Exponential size neighborhoods
Heuristic algorithms

Giovanni Righini

University of Milan
Department of Computer Science (Crema)
Very Large Scale Neighbourhood Search (VLNS)

The *steepest descent* heuristic is

- very effective when attraction basins are few and large
- little effective when attraction basins are many and small.

Larger neighborhoods

- are likely to generate larger attraction basins
- but they require more computing time to be explored.

*Very Large Scale Neighbourhood (VLSN) Search* algorithms use exponential-size neighborhoods that are visited in polynomial time.

Two strategies avoid exponential computing time:

1. the neighborhood is explored in a heuristic way, providing a good solution but not necessarily the best one in the neighborhood;
2. the neighborhood is defined in a way that allows to find the best neighbor solution in polynomial time.
Variable Depth Search (VDS)

Operations-based neighborhoods can be easily parameterized

$$N_{O_k}(x) = \{ x' \in X : x' = o_k(o_{k-1}(\ldots o_1(x))) \text{ with } o_1, \ldots, o_k \in O \}$$

The goal is

- to define a complex move as a sequence of elementary moves
- to build the sequence optimizing each elementary move
- to accept the final solution only if it improves upon the initial one.

The length $k$ of the sequence takes large values only when it is convenient: it adapts at each step.
Variable Depth Search

Given $x^{(t)}$, for each $\tilde{x} \in N(x^{(t)})$, instead of evaluating $z(\tilde{x})$

1. select the best elementary move from $\tilde{x}$ in a neighborhood $\tilde{N} \subseteq N$;
2. if it improves, execute it and go to step 1, otherwise stop;
3. keep the best solution found.

$$y^* := \tilde{x}; \quad y := \tilde{x}; \quad \text{Stop} := false;$$

$While \quad \text{Stop} = false \quad do$

$$y' := \arg \min_{x \in \tilde{N}(y)} z(x);$$

$If \quad z(y') \geq z(x^{(t)}) \quad then \quad \text{Stop} := true; \quad else \quad y := y';$

$If \quad z(y) < z(y^*) \quad then \quad y^* := y';$

$EndWhile;$
VDS vs. Steepest descent

Compared to *steepest descent*, Variable Depth Search

- finds a local optimum for each solution in the neighborhood as in a kind of one-step *look-ahead* algorithm;
- it allows for worsening solutions along the sequence of elementary moves (but not with respect to the initial solution);
- it requires some care to avoid that the elementary moves in a sequence cancel one another;
- for the sake of efficiency, it is common that
  - it use a reduced neighborhood $\tilde{N} \subset N$ to scan the elementary moves
  - it apply a first-improve strategy
    - when scanning the elementary moves at each step
    - when scanning the solution in the initial neighborhood
The Lin-Kernighan algorithm for the TSP

The neighborhood \( N_{R_k}(x) \) for the symmetric TSP contains the solutions obtained in this way:

- delete \( k \) edges of \( x \);
- insert \( k \) edges to reconstruct a Hamiltonian cycle, possibly reversing some parts of the tour.

The Lin-Kernighan algorithm applies the VDS to sequences of 2-opt exchanges: each \( k \)-opt exchange is equivalent to a sequence of \((k - 1)\) 2-opt exchanges, where each of them deletes one of the two edges inserted by the previous one.

Therefore for each exchange \((i, j) \in N_{R_k}(x)\)

- the exchanges that delete edge \((s_i, s_{j+1})\) and each other arc of \( x \) are evaluated and the best one, \((i', j')\) is selected;
- if it improves upon \( x \), the exchange \((i', j')\) is done;
- the exchanges that delete edge \((s_{i'\prime}, s_{j'\prime+1})\) and each other arc of \( x \) are evaluated;
- \( \ldots \)
- if the best solution found is better than \( x \), then it is accepted.
Example: Lin-Kernighan algorithm

The first exchange is $(1, 3)$, which reverses the path $(v_1, \ldots, v_3)$. 
Example: Lin-Kernighan algorithm

The exchange has replaced \((\nu_2, \nu_1)\) and \((\nu_3, \nu_4)\) with \((\nu_3, \nu_2)\) and \((\nu_4, \nu_1)\).

Now we scan the exchanges that delete \((\nu_4, \nu_1)\) and another edge.
Example: Lin-Kernighan algorithm

The best among them replaces \((v_4, v_1)\) and \((v_5, v_6)\) with \((v_5, v_4)\) and \((v_6, v_1)\).

Now we scan the exchanges that delete \((v_6, v_1)\) and another edge.
The best among them replaces \((v_6, v_1)\) and \((v_8, v_7)\) with \((v_6, v_7)\) and \((v_8, v_1)\).

Now we scan the exchanges that delete \((v_8, v_1)\) and another edge.
Example: Lin-Kernighan algorithm

The best among them replaces \((v_8, v_1)\) and \((v_{10}, v_9)\) with \((v_9, v_8)\) and \((v_{10}, v_1)\).

Now we scan the exchanges that delete \((v_{10}, v_1)\) and another arc.
Example: Lin-Kernighan algorithm

The best among them replaces \((v_{10}, v_1)\) and \((v_{12}, v_{11})\) with \((v_{11}, v_{10})\) and \((v_{12}, v_1)\).

Now we scan all exchanges that delete \((v_{12}, v_1)\) and another edge: since all of them produce solutions that are worse than the initial one, the algorithm stops.
Implementation details

- In order to avoid to undo previous moves, the second edge that is deleted at each iteration must belong to the initial solution;
- this implies an upper bound to the length of the sequence;
- it is possible to prove that stopping the sequence as soon as the exchanges do not improve the initial solution does not worsen the result:
  - the overall change in the objective is the sum of the variations due to the exchanges

\[ \delta (x, o_1, \ldots, o_k) = \sum_{\ell=1}^{k} \delta (x, o_\ell) \]

- any sequence of numbers with negative sum allows for a cyclic permutation whose partial sums are all negative
- hence there exists a cyclic permutation of the moves \( o_1, \ldots, o_k \) that is improving at each step

\[ \delta (x, o_1, \ldots, o_k) < 0 \Rightarrow \exists h : \sum_{\ell=1}^{\ell} \delta (x, o_{h+1}, \ldots, o_{h+\ell}) < 0 \ \forall \ell = 1, \ldots, k \]
**Destroy-and-repair methods**

Every exchange from solution $x$ to solution $x'$ can be seen as
- an addition of subset $A = x' \setminus x$ to $x$;
- a deletion of subset $D = x \setminus x'$ from $x$.

Insertions and deletions produce exchanges, but
- the subsets $x \cup \{j\} \setminus \{i\}$ obtained by exchanging single elements could be infeasible or very bad
- enlarging the neighborhood to exchanges of several elements could be inefficient
- in many problems $A$ and $D$ could have different cardinality, because the cardinality of the solutions is not fixed. (e.g., KP, SCP...)

An alternative is
- to delete a subset $D \subset x$ of cardinality $\leq k$ and to complete it with a constructive heuristic (especially if $x \setminus D \in X$ for any $D$ (e.g. the KP));
- to add a subset $A \subset E \setminus x$ of cardinality $\geq k$ and to prune it with a destructive heuristic (especially if $x \cup A \in X$ for any $A$ (e.g. the SCP)).
**Destroy-and-repair methods**

The complexity decreases from $O(n^{|A|} n^{|D|} \gamma(n))$, where
- the number of subsets that can be inserted is $O(n^{|A|})$;
- the number of subsets that can be deleted is $O(n^{|D|})$;
- $\gamma(n)$ is the complexity of evaluating the feasibility and the objective;

the complexity decreases to $O(n^{|D|} T_{\text{constr}}(n))$ or $O(n^{|A|} T_{\text{destr}}(n))$, where
- $T_{\text{constr}}(n)$ is the complexity of the constructive heuristic;
- $T_{\text{destr}}(n)$ is the complexity of the destructive heuristic.

Furthermore, one can use all already mentioned techniques to reduce the complexity:
- to concentrate exchanges on elements with low or high cost
- to employ first-improve instead of best-improve strategy
- to insert random steps
- to exploit memory, by favoring or forbidding exchanges of elements used often or rarely in the previous iterations.
Another set of methods is based on neighborhoods whose optimal solution can be found by solving a polynomial-time optimization problem (usually defines on a suitable graph or matrix):

- **packing**: Dynasearch;
- **negative cost cycle**: cyclic exchanges;
- **shortest path**: ejection chains, order-and-split.

Such matrices or graphs are defined as improvement matrices or graphs.
Dynasearch

The variation $\delta(x, o)$ of the objective function due to an elementary move $o \in O$ often depends only on a small part of the solution.

Operations $o'$ that act on other parts of the solution have an independent effect: the order of the operations is not important.

$$z(o'(o(x))) = z(o(o'(x)))$$

If $z(\cdot)$ is additive, the effects of the two exchanges simply add up.

Example:
- exchanges between different branches for the CMSTP or different routes for the VRP
- 2-opt exchanges for the TSP on disjoint parts of the cycle
Dynasearch

The composite move is a set of elementary moves as in VDS, but the elementary moves must have mutually independent effects on feasibility and the objective.

It can be modeled by an improvement matrix $A$ where

- the rows represent the components of the solution (e.g., branches in the CMSTP, routes in the VRP, parts of the cycle in the TSP)
- the columns represent the elementary moves: each column has a value equal to the improvement $-\delta$ it provides
- if move $j$ affects component $i$, then $a_{ij} = 1$, otherwise $a_{ij} = 0$.

One wants to determine an optimal packing of the columns, that is the maximum value subset of columns that do not cover any row more than once.
Dynasearch

The *Set Packing Problem* is in general $\mathcal{NP}$-hard, but

- it is polynomially solvable on particular matrices (e.g. the *TSP*)
- if every move affects at most two components
  - rows correspond to vertices
  - columns correspond to edges
  - each packing of the columns corresponds to a matching on the corresponding graph

and the graph is bipartite.
Cyclic exchanges

In many problems

- the feasible solutions partition the elements into components $S_\ell$ (branches, routes, bins, ...)
- the feasibility only depends on each single component
- the objective function is additive with respect to the components

\[ z(x) = \sum_{\ell=1}^{r} z(S_\ell) \]

For these problems it is natural to define the set of operations $T_k$ containing the transfers of $k$ elements from their component to another. From $T_k$ the corresponding neighborhood $N_{T_k}$ derives.

The constraints often forbid simple transfers but the number of multiple transfers grows rapidly with $k$.

One wants to define a suitable subset of $N_{T_k}$ that can be explored efficiently.
The improvement graph

The improvement graph allows to describe sequences of transfers

- a node $i$ corresponds to an element $i$ of the ground set $E$
- an arc $(i, j)$ corresponds to
  - transferring element $i$ from its current component $S_i$ to the current component $S_j$ of element $j$
  - deleting element $j$ from $S_j$;
- the cost $c_{ij}$ of the arc corresponds to the variation of the cost of the component $S_j$ in the objective function:
  \[
  c_{ij} = z(S_j \cup \{i\} \setminus \{j\}) - z(S_j)
  \]
  with $c_{ij} = +\infty$ if it is infeasible to transfer $i$ deleting $j$.

A cycle in such a graph corresponds to a closed sequence of transfers.

The cost of the cycle corresponds to the cost of the sequence

- but this is guaranteed only if each node is in a different component.

The algorithm searches for the minimum cost cycle satisfying this condition.
Example: the CMSTP

Composite move:
- node 4 goes from the red branch to the blue branch
- node 3 goes from the blue branch to the green branch
- node 11 goes from the green branch to the brown branch
- node 8 goes from the brown branch to the red branch
Searching for the minimum cost cycle

Owing to the constraint that each component must be affected only once, the problem of finding the minimum cost cycle is $\mathcal{NP}$-hard, but

- the same constraint allows for rather efficient dynamic programming algorithms
  
  *(the cycle has no more than $r$ arcs, where $r$ is the number of components)*

- one can discard all paths with non-negative cost

- there are polynomial time algorithms to compute
  
  - unconstrained negative cost cycles (Floyd-Warshall)
  
  - minimum mean cost cycles.

Although they do not respect the constraint on the components, they provide

- a lower bound that can prove that no negative cost cycle exists

- a negative cost cycle that can comply with the constraint just by chance or a cycle that can be modified to obtain a feasible one;

- we can search for the negative cost cycle with a polynomial time heuristic.
Non-cyclic sequences of exchanges

It is possible to generate **non-cyclic sequences of transfers** (so that the cardinality of the components can change).

It is enough to **modify the improvement graph** as follows:
- to insert a source node;
- to insert a node for each component;
- to insert arcs from the source node to the element nodes;
- to insert arcs from the element nodes to the component nodes.

Now we search for the **minimum cost path**
- from the source node to a component node
- visiting only one component node (guaranteed by construction)
- such that if it visits a component node it does not visit any element node associated with it

These paths correspond to open sequences of transfers where
- one component looses an element;
- zero or more components loose an element and receive another one;
- one component receives an additional element.
Example: the CMSTP

(4, Rosso, Blu) (3, Blu, Verde) (11, Verde, Marrone)

sorgente
nodи (sottografo completo)
Order-first split-second

The method *Order-first split-second* for partitioning problems

- builds an initial permutation of the elements
- partitions the sequence optimally into components with the additional constraint that elements in the same component must be consecutive in the permutation.

Obviously, the solution depends on the initial permutation: it is reasonable to compute the solution for different permutations generating a two levels methods:

1. at the upper level, we modify the permutation
2. at the lower level, we optimize the partition

*Problem: there are more permutations than solutions: the upper level must guarantee that the solution at the lower level change.*
The auxiliary graph

Also in this case one exploits an auxiliary graph.

Given the permutation \((s_1, \ldots, s_n)\) of the elements of the ground set \(E\)

- each node \(s_i\) corresponds to an element \(s_i\) of \(E\);
- there is an additional dummy node \(s_0\);
- each arc \((s_i, s_j)\) with \(i < j\) corresponds to a potential component \(S_\ell = (s_{i+1}, \ldots, s_j)\) formed by the element of the permutation from \(s_i\) excluded to \(s_j\) included;
- the cost \(c_{s_i, s_j}\) corresponds to the cost \(z(S_\ell)\) of the component;
- the arc does not exist if the component is infeasible.

Then:

- every path from \(s_0\) to \(s_n\) represents a partition of \(E\);
- the cost of the path coincides with the cost of the partition
- the graph is acyclic: finding the optimal path costs \(O(m)\) where \(m \leq n(n - 1)/2\) is the number of arcs.
Example: the VRP

Given an instance of the VRP with 5 nodes and capacity \( W = 10 \)

the arcs corresponding to infeasible routes (weight > \( W \)) do not exist, and the costs are those of routes \((d, s_{i+1}, \ldots, s_j, d)\).
Example: the VRP

The optimal path corresponds to three routes: \((d, s_1, s_2, d)\), \((d, s_3, d)\) and \((d, s_4, s_5, d)\).