

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>

Very Large Scale Neighbourhood Search

Larger neighbourhoods yield in general larger attraction basins, so that

- the *steepest descent* heuristic becomes very effective
- but the exploration time is longer

The *Very Large Scale Neighbourhood* (*VLSN*) *Search* approaches have

- neighbourhoods exponential in $|B|$ (or high-order polynomial)
- explored in low-order polynomial time

Two strategies allow to limit the computational time

- 1 select a neighbourhood in which the objective can be optimized without an exhaustive exploration
- 2 explore the neighbourhood heuristically and return a promising neighbour solution, instead of the best one

Efficient visit of exponential neighbourhoods

Some methods are based on **neighbourhoods** whose optimal solution can be found solving a problem on an auxiliary matrix or graph

- **set packing**: *Dynasearch*
- **negative cost circuit**: cyclic exchanges
- **shortest path**: *ejection chains, order-and-split*

Such tools are usually defined **improvement matrices** or **graphs**

The variation $\delta f_o(x)$ of the objective function implied by an operation $o \in \mathcal{O}$ often depends only on a part of solution x

Operations modifying the same parts are in general incompatible

Operations modifying different parts have independent effects and the order in which they are performed is irrelevant

$$f(o'(o(x))) = f(o(o'(x)))$$

If $f(\cdot)$ is additive, the two exchanges simply sum their effects

Examples:

- exchanges between different branches for the *CMSTP* or different circuits for the *VRP* (*otherwise, the exchange could be unfeasible*)
- 2-opt exchanges for the *TSP* operating on disjoint segments of the circuit (*otherwise, one of the two arcs is removed or reversed*)



A whole set of moves can be performed all together summing their effects

Let a **composite move** be a **set of elementary moves** with **mutually independent effects** on feasibility and the objective

The situation can be modelled with an **improvement matrix A** in which

- the rows represent the components of the solution (e.g., branches in the *CMSTP*, circuits in the *VRP*, circuit segments in the *TSP*)
- the columns represent the elementary moves:
each column has a value equal to the objective improvement $-\delta f$
- if the move j impacts on component i , $a_{ij} = 1$; otherwise $a_{ij} = 0$

Determine the **optimum packing of the columns**, that is the **subset of nonconflicting columns of maximum value**

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

- on special matrices it is polynomial (as in the matrix from the *TSP*)
- if each move modifies at most two components
 - the rows can be seen as nodes
 - the columns can be seen as edges
 - each packing of columns becomes a matching

and the maximum matching problem is polynomial

Cyclic exchanges

In many problems

- every feasible solution partitions objects into components $S^{(\ell)}$:
it consists of assignments of objects to components (i, S_i)
(vertices or edges into branches for the CMSTP, nodes or arcs into circuits for the VRP, objects into containers in the BPP, etc...)
- the feasibility is associated to the single components
- the objective function is additive with respect to the components

$$f(x) = \sum_{\ell=1}^r f(S^{(\ell)})$$

It is natural to define for these problems the set of operations \mathcal{T}_k which includes the transfers of k elements from their component to another and to derive from \mathcal{T}_k the neighbourhood $N_{\mathcal{T}_k}$

- often the feasibility constraints forbid the simple transfers
- but the number of multiple transfers quickly grows with k

We want to find a subset of $N_{\mathcal{T}_k}$ large, but efficient to explore

The improvement graph

The **improvement graph** allows to describe sequences of transfers

- a **node** i corresponds to an **element** i of the current solution x
- an **arc** (i, j) corresponds to
 - the **transfer of element** i from its current component S_i to the current component S_j of element j
 - the **deletion of element** j from component S_j
- the **cost of arc** c_{ij} corresponds to the **variation of the contribution of** S_j to the objective

$$c_{ij} = f(S_j \cup \{i\} \setminus \{j\}) - f(S_j)$$

with $c_{ij} = +\infty$ if it is **infeasible to transfer** i deleting j

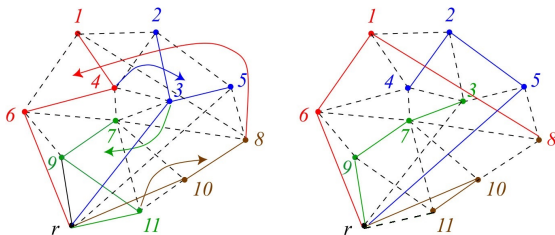
A circuit in such a graph corresponds to a closed sequence of transfers

The cost of the circuit corresponds to the cost of the sequence

- but only **if each node belongs to a different component**

Find the minimum cost circuit satisfying this condition

Example: the *CMSTP*



Consider the composite move $(4, 3)$, $(3, 11)$, $(11, 8)$, $(8, 4)$:

- vertex 4 moves into the blue branch to replace vertex 3
- vertex 3 moves into the green branch to replace vertex 11
- vertex 11 moves into the brown branch to replace vertex 8
- vertex 8 moves into the red branch to replace vertex 4

The cost variation for subtree S_j yields the cost of arc c_{ij}

The weight of branch S_j varies by $w_i - w_j$: if unfeasible, forbid the arc

Search for the minimum cost circuit (1)

The problem is actually \mathcal{NP} -hard, but

- the constraint of visiting only once each component allows a rather efficient **dynamic programming algorithm** that grows partial paths
(if the components are r , the circuit has at most r arcs)
- **all partial paths of cost ≥ 0 can be neglected** because
 - the total variation of the objective sums the effect of the single moves

$$\delta f_{o_1, \dots, o_k}(x) = \sum_{\ell=1}^k \delta f_{o_\ell}(x)$$

- every sequence of numbers with negative sum admits a cyclic permutation whose partial sums are all negative
- therefore, there is a cyclic permutation of the moves o_1, \dots, o_k with $\delta f < 0$ at each step

$$\delta f_{o_1, \dots, o_k}(x) < 0 \Rightarrow \exists h : \sum_{\ell=1}^{\ell} \delta f_{o_{h+1}, \dots, o_{h+\ell}}(x) < 0 \quad \forall \ell = 1, \dots, k$$

Search for the minimum cost circuit (2)

Moreover,

- the problem admits heuristic polynomial algorithms
 - there are polynomial algorithms to compute
 - a nonminimum negative circuit (Floyd-Warshall), if any exists
 - the minimum average cost circuit (total cost / number of arcs)
- relaxing the constraint on the components; if such solutions have
- positive cost, they prove that no negative circuit exists
 - negative cost, they provide a **relaxed solutions** that can be
 - optimal (if feasible by chance)
 - potentially modified to obtain feasible heuristic solutions

Noncyclic exchange chains

It is also possible to create **noncyclic transfer chains**, so that the cardinality of the components can vary

It is enough to **add to the improvement graph**

- a source node
- a node for each component
- arcs from the source node to the nodes associated to the elements
- arcs from the nodes associated to the elements to the nodes associated to the components

Then, find the **minimum cost path** that

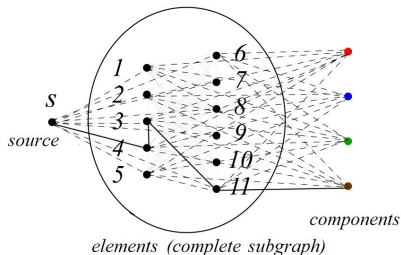
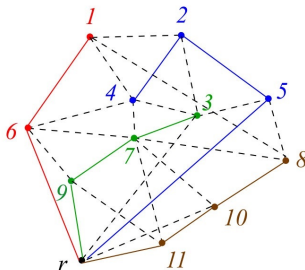
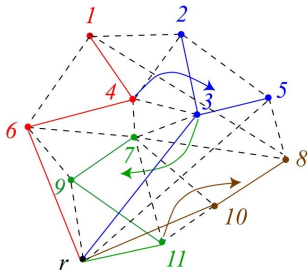
- starts from the source node
- ends in a component node (guaranteed by the topology)
- never visits two nodes associated to the same component

These paths correspond to open transfer chains in which

- a component loses an element
- zero or more components lose an element and acquire another one
- a component acquires an element

Example: the *CMSTP*

Noncyclic exchange $(s, 4)$, $(4, 3)$, $(3, 11)$, $(11, S_4)$



Order-first split-second

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

- builds a starting permutation of the elements
- partitions the elements into components in an optimum way under the additional constraint that elements of the same component be consecutive in the starting permutation

Of course, the solution depends on the starting permutation:
it is reasonable to repeat the resolution for different permutations
creating a two-level method

- 1 the upper level selects a permutation
- 2 the lower level computes the optimal partition for the permutation

*Problem: different permutations yield the same solution
(the permutations are more numerous than the solutions)*

The auxiliary graph

Once again, we exploit an auxiliary graph

Given the permutation (s_1, \dots, s_n) of the elements

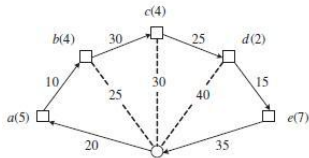
- each node s_i corresponds to a element s_i
plus a fictitious node s_0
- each arc (s_i, s_j) with $i < j$ corresponds to a potential component $S_\ell = (s_{i+1}, \dots, s_j)$ formed by the elements of the permutation
 - from s_i excluded
 - to s_j included
- the cost c_{s_i, s_j} corresponds to the cost of the component $f(S_\ell)$
- the arc does not exist if the component is unfeasible

Consequently

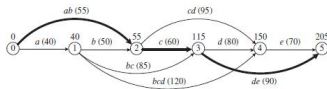
- each path from s_0 a s_n represents a solution (partition of elements)
- the cost of the path coincides with the cost of the partition
- the graph is acyclic: finding the optimum path costs $O(m)$ where $m \leq n(n-1)/2$ is the number of arcs

Example: the *VRP*

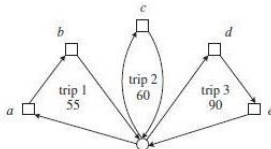
Given an instance of *VRP* with 5 nodes and capacity $W = 10$



the arcs corresponding to unfeasible paths (weight $> W$) do not exist, the costs of the arcs are the costs of the *TSP* solutions for $\{d, s_{i+1}, \dots, s_j\}$



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



Variable Depth Search (VDS)

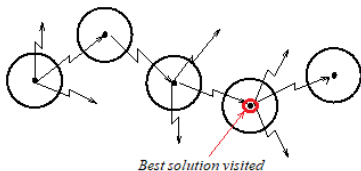
The neighbourhoods based on operations can be easily parameterised

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

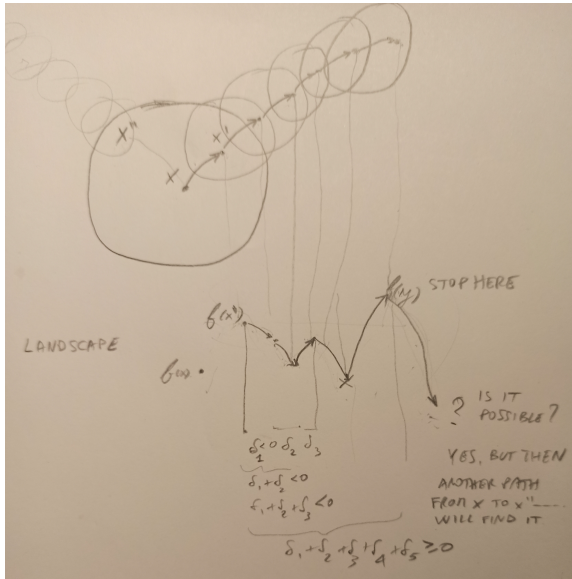
but it would be nice to tune the number of operations k as required

The idea is to define a **composite move** as **sequence of elementary moves**

- consider each solution x' in the basic neighbourhood $N_{\mathcal{O}_1}(x)$
- from it, make a sequence of moves **optimizing each elementary step**, but **allowing worsening moves** and **forbidding backward moves**
- **terminate** when the current solution y becomes worse than x' or all moves are forbidden (**the length k of the sequence is variable**)
- **return the best solution y^* found along the sequence**



Variable Depth Search



Scheme of the Variable Depth Search

Given $x^{(t)}$, for each $x' \in N(x^{(t)})$, instead of evaluating only $f(x')$

- ① find a promising solution \tilde{y} in a neighbourhood $\hat{N}(x') \subseteq N(x')$
- ② as long as \tilde{y} improves $x^{(t)}$, replace x' with \tilde{y} and go to 1
- ③ return the best solution y^* found during the whole process

For each $x' \in N(x)$

{ *Steepest descent* }

Compute $f(x')$

{ *Variable Depth Search* }

$y := x'$; $y^* := x'$; Stop := false;

While Stop = false do

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

If $f(\tilde{y}) \geq f(x')$ then Stop := true; else $y := \tilde{y}$;

If $f(\tilde{y}) < f(y^*)$ then $y^* := \tilde{y}$;

EndWhile;

Compute $f(y^*)$;

It is a sort of roll-out mechanism for exchange algorithms

Differences with respect to *steepest descent*

With respect to *steepest descent* exploration

- *VDS* finds a local optimum for each solution of the neighbourhood performing a sort of one-step *look-ahead*
- *VDS* admits worsenings along the sequence of elementary moves (but never with respect to the starting solution)
- *VDS* makes moves that increase the distance from the starting point to avoid cyclic behaviours (*gradually restricting the neighbourhood*)

In order to limit the computational effort

- the elementary moves use a reduced neighbourhood $\hat{N} \subseteq N$
- \hat{N} (elementary step) is explored with the *first-best strategy*
- N (basic neighbourhood) is explored with the *first-best strategy*

Lin-Kernighan's algorithm for the symmetric TSP

Neighbourhood $N_{\mathcal{R}_k}(x)$ includes the solutions obtained

- deleting k arcs of x
- adding other k arcs that recreate a Hamiltonian circuit
- possibly inverting parts of the circuit *(leaving the cost unchanged)*

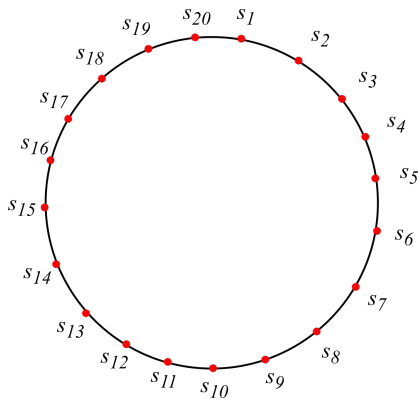
Lin-Kernighan's algorithm is a VDS with sequences of 2-opt exchanges:
a k -opt exchange is equivalent to a sequence of $(k - 1)$ 2-opt exchanges,
where each deletes one of the two arcs added by the previous exchange

Then for each solution $x' \in N_{\mathcal{R}_2}(x)$ obtained by exchange (i, j)

- evaluate the 2-opt exchanges that delete the added arc (s_i, s_{j+1}) and each arc of $x \cap x'$ to find the best exchange (i', j')
- if this improves upon x , perform exchange (i', j') , obtaining x''
- evaluate the exchanges that delete $(s_{i'}, s_{j'+1})$ and each arc of $x \cap x'' \dots$
- \dots
- if the best solution among x', x'', \dots is better than x , accept it

Example: Lin-Kernighan's algorithm

Explore all the solutions $x' \in N_{\mathcal{R}_2}(x)$, obtained with exchanges (i, j)

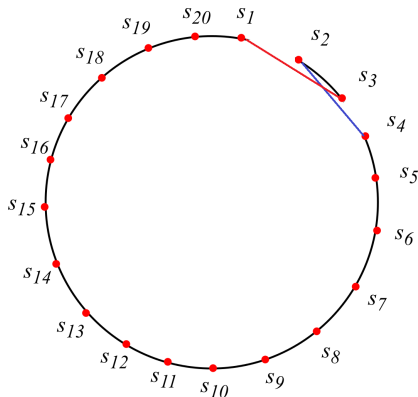


$$x = (s_1 \overline{s_2 s_3} s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20})$$

Let us focus on the exchange $(1, 3)$, that reverts (s_2, \dots, s_3)

Example: Lin-Kernighan's algorithm

The exchange $(1, 3)$ replaces (s_1, s_2) and (s_3, s_4) with (s_1, s_3) and (s_2, s_4)

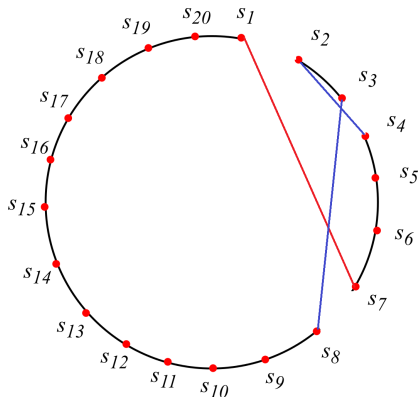


$$x' = (s_1 \overline{s_3 s_2 s_4 s_5 s_6 s_7} s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20})$$

Search for the best exchange that removes (s_1, s_3) and an arc of $x \cap x'$
Let us suppose that it is $(1, 7)$, which reverts (s_3, \dots, s_7)

Example: Lin-Kernighan's algorithm

The exchange (1, 7) replaces (s_1, s_3) and (s_7, s_8) with (s_1, s_7) and (s_3, s_8)

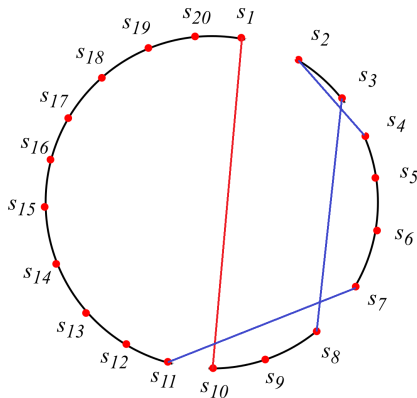


$$x'' = (s_1 \overline{s_7 s_6 s_5 s_4 s_3 s_8 s_9 s_{10}} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20})$$

Search for the best exchange that removes (s_1, s_7) and an arc of $x \cap x''$
Let us suppose that it is (1, 10), which reverts (s_7, \dots, s_{10})

Example: Lin-Kernighan's algorithm

The exchange $(1, 10)$ replaces (s_1, s_7) and (s_{10}, s_{11}) con (s_1, s_{10}) and (s_7, s_{11})

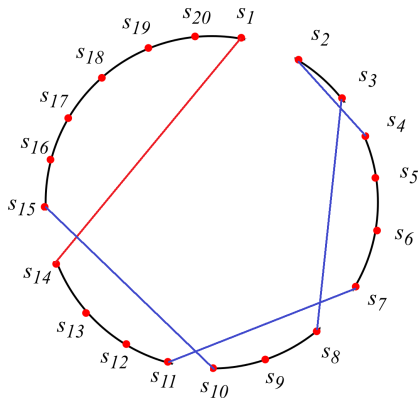


$$x''' = (s_1 \overline{s_{10} s_9 s_8 s_3 s_2 s_4 s_5 s_6 s_7 s_{11} s_{12} s_{13} s_{14}} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20})$$

Search for the best exchange that removes (s_1, s_{10}) and an arc of $x \cap x'''$
 Let us suppose that it is $(1, 14)$, which reverts (s_{10}, \dots, s_{14})

Example: Lin-Kernighan's algorithm

The exchange $(1, 14)$ replaces (s_1, s_{10}) and (s_{14}, s_{15}) con (s_1, s_{14}) and (s_{10}, s_{15})

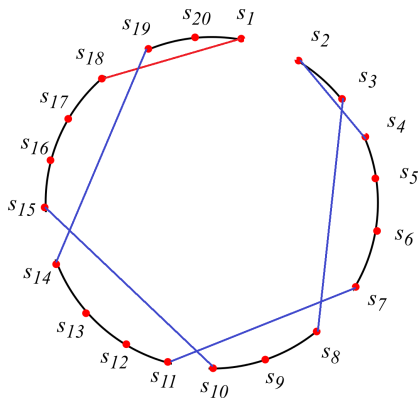


$$x^{iv} = (s_1 \overline{s_{14} s_{13} s_{12} s_{11} s_7 s_6 s_5 s_4 s_2 s_3 s_8 s_9 s_{10} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20}})$$

Search for the best exchange that removes (s_1, s_{14}) and an arc of $x \cap x^{iv}$
 Let us suppose that it is $(1, 18)$, which reverts (s_{14}, \dots, s_{18})

Example: Lin-Kernighan's algorithm

The exchange $(1, 18)$ replaces (s_1, s_{14}) and (s_{18}, s_{19}) con (s_1, s_{18}) and (s_{14}, s_{19})



$$x^v = (s_1 \ s_{18} \ s_{17} \ s_{16} \ s_{15} \ s_{10} \ s_9 \ s_8 \ s_3 \ s_2 \ s_4 \ s_5 \ s_6 \ s_7 \ s_{11} \ s_{12} \ s_{13} \ s_{14} \ s_{19} \ s_{20})$$

Search for the best exchange that removes (s_1, s_{18}) and an arc of $x \cap x^v$
Let us suppose that all exchanges yield solutions worst than x :
terminate, returning the best solution found

Implementation details

- each step deletes an arc of the starting solution to avoid going back and one of the arcs added in the previous step to reduce complexity
- this imposes an upper bound on the length of the sequence
- stopping the sequence as soon as the solution is no longer better than the starting solution does not impair the result
 - the total variation of the objective sums the effect of the single moves

$$\delta f_{o_1, \dots, o_k}(x) = \sum_{\ell=1}^k \delta f_{o_\ell}(x)$$

- every sequence of numbers with negative sum admits a cyclic permutation whose partial sums are all negative
- therefore, there is a cyclic permutation of the moves o_1, \dots, o_k with $\delta f < 0$ at each step

$$\delta f_{o_1, \dots, o_k}(x) < 0 \Rightarrow \exists h : \sum_{\ell=1}^{\ell} \delta f_{o_{h+1}, \dots, o_{h+\ell}}(x) < 0 \quad \forall \ell = 1, \dots, k$$

Iterated greedy methods (*destroy-and-repair*)

Every exchange can be seen as a combination of addition and deletion

$$x' = x \cup A \setminus D$$

with $A = x' \setminus x$ and $D = x \setminus x'$

However

- single swaps $x' = x \cup \{j\} \setminus \{i\}$ can give bad or unfeasible results
- larger neighbourhoods can be inefficient
- in many problems the right cardinalities of A and D are unknown, because the solutions have nonuniform cardinality (e.g., *KP*, *SCP*...)

A possible idea is to

- delete from x a subset $D \subset x$ of cardinality $\leq k$ and complete it with a constructive heuristic

or

- add to x a set $A \subset B \setminus x$ of cardinality $\leq k$ and reduce it with a destructive heuristic

Selection of A and D

Even if one of the two subsets is build with a polynomial heuristic, considering all possible choices for the other subset is usually not viable

- the number of the possible subsets to add is $O(n^{|A|})$
- the number of the possible subsets to delete is $O(n^{|D|})$

Therefore, the complexity is further reduced

- generating A (or D) with randomized heuristics, biased to select promising elements based on their cost/value
- applying the first-best strategy

They are metaheuristics, rather than heuristics