OPTIMIZATION ALGORITHMS

UNIVERSITÀ DEGLI STUDI DI BRESCIA

Local Search Methods

Prof. **Renata Mansini**

Academic year 2021/2022

Research Group MAO@DII

Models and Algorithms for Optimization at Department of Information Engineering

Sito web: `http://or-dii.unibs.it/`

Heuristic Algorithms 1/2

From the greek word εὑρίσκω, *eurisko* → to find or discover.

Objective:

determining a good solution (not necessarily the optimal one) in a reasonable amount of time.

Properties:

- Computing time should not grow too rapidly when the size of the problem grows (i.e. time complexity should be a polynomial function in the size of the problem with low degree);
- Identified solutions should be, at least for the majority of the instances of the problem, optimal or close to the optimal one;
- It is a compromise between quality of the solution and time needed to find it.

Heuristic Algorithms 2/2

Minimum Problem

A heuristic A determines an **Upper Bound** on the optimal value:

$$z^* : z^A \geq z^*$$

(**Lower Bound** for a **maximum** problem).

Two main families of heuristic algorithms:

- **Constructive heuristic algorithms**
  - start from an empty solution;
  - iteratively determine new elements to add, until a complete (feasible) solution has been reached.
- **Local search algorithms**
  - start from a feasible solution x;
  - iteratively try to improve it by applying changes to the current solution (*moves* in a neighborhood of x);
  - terminate when there are no moves (of the selected type) that can improve the current solution (local minimum).

Examples of constructive algorithms: TSP 1/3

The Traveling Salesman Problem (TSP):

The constructive algorithms for the TSP start from an empty solution and iteratively determine the new vertices to add to the solution, until a complete solution has been reached:

1. Chose an arbitrary vertex and consider the corresponding self-loop as the initial partial solution S.
2. Expand S by inserting one by one the remaining vertices, finally obtaining an hamiltonian cycle.

   This procedure is done in two separate phases:

   a) select a vertex k, not included in S, according to some ***criterion***;
   b) insert vertex k in S, between two specific consecutive vertices u, v.

Examples of constructive algorithms: TSP 2/3

Several selection ***criteria*** can be used in point 2a:

- *Nearest neighbor*: select the vertex with the minimum distance from S. (average case: 20%; worst case: error equal to 1; computational complexity $O(n^2)$)
- *Farthest insertion*: select the vertex with the maximum distance from S. (average case: 10%; worst case: error equal to $\lceil logn \rceil + 1$; computational complexity $O(n^2)$)
- *Arbitrary insertion*: randomly select the vertex to insert in S. (average case: 11%; worst case: error equal to $\lceil logn \rceil + 1$; computational complexity $O(n^2)$).
- *Cheapest insertion*: select the vertex with the minimum insertion cost. (average case: 17%; worst case: error equal to 1; computational complexity $O(n^2 logn)$).

Examples of constructive algorithms: TSP 3/3

Point 2b: the vertex is usually inserted in a way that minimizes the "insertion cost":

- for each $(i, j) \in S$ we compute $d_{ij} = c_{ik} + c_{kj} - c_{ij}$;
- k is inserted between the consecutive vertices u, v such that:

$$d_{uv} \leq d_{ij} \qquad \forall (i, j) \in S.$$

Examples of constructive algorithms: 0–1 KP

0–1 Knapsack Problem

Greedy Algorithm:

1. Sort the items in non-increasing order of profit/weight ratio;
2. Add items in the knapsack until you reach the first one that exceeds the capacity.

Alternatively: sort the items in non-increasing order of profit or non-decreasing order of weight.

Algorithm performance analysis

How do you evaluate the quality of an algorithm?

There are two ways to analyze the performance of a heuristic algorithm:

1. *Experimental.* The quality of the solutions obtained for a benchmark set of instances is evaluated:
   – Can always be performed.
   – Not generalizable.
   – Performance are evaluated according to the solution quality/computing time ratio.
2. *Worst case.* The greatest error from the optimal solution is analytically computed:
   – Generalizable, but hard to determine.

Greedy Algorithm for the TSP

1. **Step 1.** Choose a starting node i and add it to the empty partial solution W, i.e. $W := \{i\}$. Set $r = i$.
2. **Step 2.** Select $s \in V \setminus W$ with minimum distance from node r, i.e. $d_{rs} = \min_{j \in V \setminus W} d_{rj}$, where d_{ij} is the length of arc (i, j).
3. **Step 3.** Set $W := W \cup \{s\}$ and $r = s$. If $W = V$, stop. Node i is the one following node s, and W is now a cycle. If that is not the case, return to Step 2.

The heuristic has a polynomial computational complexity: the number of operations performed by the algorithm is $O(n^2)$, where $n = |V|$.

The quality of the solution can be terrible: the ratio between the identified solution and the optimal solution can be infinitely high. Let's see it in the next example.

Greedy Algorithm for the TSP: worst case example 1/2

Let us consider an undirected complete graph with 4 nodes. The following (symmetric) table shows the distances between each pair of nodes:

	1	2	3	4
1	-	2	M	2
2	2	-	2	1
3	M	2	-	2
4	2	1	2	-

Starting from node 1 (and similarly for all the other nodes), we obtain the hamiltonian cycle: $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ of length $5 + M$.

Greedy Algorithm for the TSP: worst case example 2/2

- For $M \leq 3$, the cycle identified by the heuristic is the optimal one, but for $M > 3$ the optimal solution becomes the hamiltonian cycle: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$, with objective function value equal to 8.
- For $M > 3$ the ratio between the objective function value of the cycle identified by the heuristic and the optimal one is:

$$\frac{5 + M}{8}$$

and it grows to infinity for $M \rightarrow \infty$.

Neighborhood Function

- Given an optimization problem $P = (f, S)$
    - S set of all the feasible solutions for P;
    - $f : S \to R$ objective function;
- Neighborhood
  $N : S \to 2^{|S|}$ that $\forall i \in S$ defines $N(i) \subseteq S$ set of all the solutions close to i;
- Local Search algorithms try to improve a solution by exploring one of its neighborhoods.

Iterative Improvement Algorithms 1/3

First improvement: the neighbors of the current solution are analyzed in random order. A move is made towards the first one that improves the current solution (the algorithm exits the **for** cycle at the first improvement).

Procedure FI_Simple_Descent(s) */*$s \in S$ initial sol.*/*

Found := TRUE;
while Found = **TRUE do**
    Found := FALSE;
    for each $s' \in N(s)$ **do**
        if $f(s') < f(s)$ **then**
            $s := s'$; Found := TRUE; **break**;
end while
return (s);

Converges to a local optimum s with respect to $N(.)$, i.e. a solution $s : f(s) \leq f(i)\ \forall i \in N(s)$;

Iterative Improvement Algorithms 2/3

Best improvement: all the neighbors of the current solution are examined, and a move towards the best one is made. It is a *steepest descent* method: the move that produces the greatest improvement is made.

Procedure BI_Simple_Descent(s) */*$s \in S$ initial sol.*/*

Found := TRUE;
while Found = **TRUE do**
  Found := FALSE; s_{best} := s;
  for each $s' \in N(s)$ **do**
    if $f(s') < f(s_{best})$ **then**
      $s_{best} := s'$;
  if $s_{best} \neq s$ **then**
    $s := s_{best}$; Found := TRUE;
end while
return (s);

Iterative Improvement Algorithms 3/3

Main drawback of the Best Improvement algorithm:

Each iteration of the algorithm requires a lot of time: the time needed to evaluate $f(s)$ for $O(|N(s)|)$, where $|N(s)| :=$ is the cardinality of the neighborhood.

$$\Downarrow$$

the number of iterations required to reach the local optimum could be very high (although polynomial).

Local Search Algorithms

Advantages

- broad applicability;
- flexibility with respect to changes to the problem;
- can be used even when the solution is not feasible.

Disadvantages

- they cannot escape local minima, since they do not allow moves towards solutions that are worse than the current one.

Require

- A solution evaluator (objective function);
- feasibility check for a solution;
- neighborhood function;
- efficient neighborhood exploration technique.

Example 1:

2-opt heuristic

Traveling Salesman Problem TSP $(G(V, E), c, min)$.

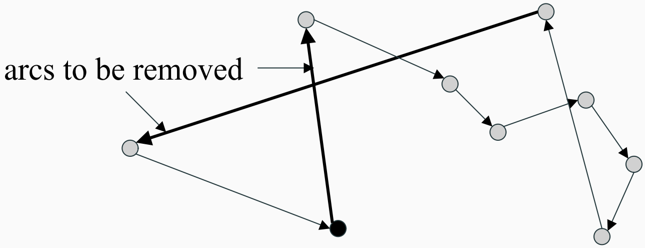
Basic idea: 2 arcs are removed at each step, then the two paths are reconnected with two different arcs.

2-opt algorithm:

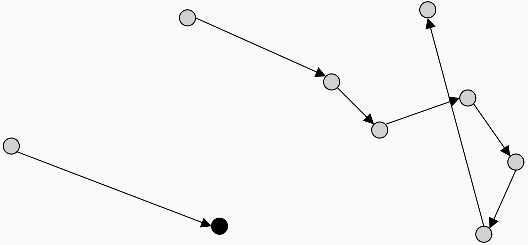
- start from an hamiltonian cycle,
- perform the 2-opt swaps between all pairs of arcs that reduce the circuit length.

Effectiveness: 8% more than the minimum, on average.

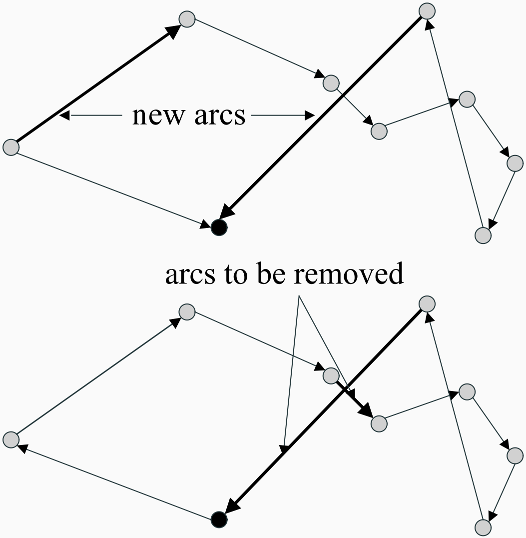
Initial cycle:



Paths:

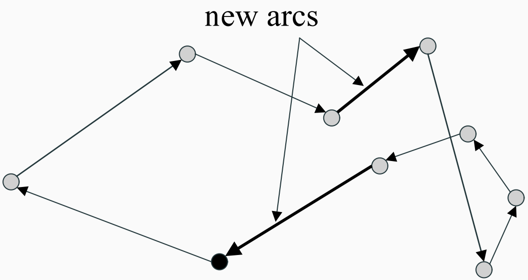


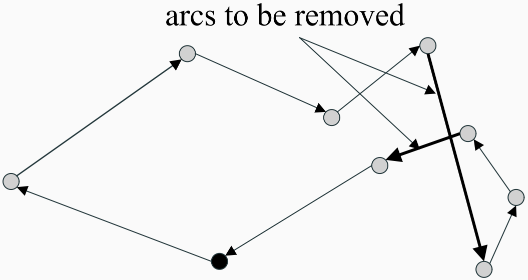
Cycle at Step 1:



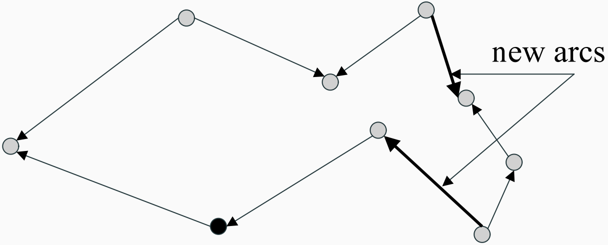
Notice the inversion of the direction of travel in one of the paths.

Cycle at Step 2:





Cycle at Step 3:



Notice the inversion of the direction of travel in one of the paths.

Example 2:

3-opt heuristic

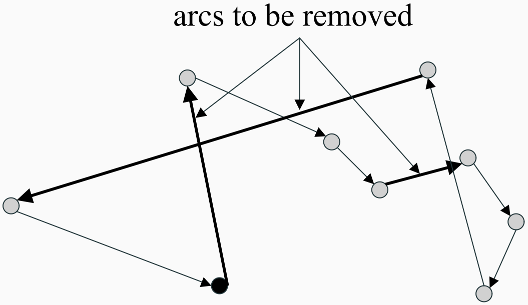
Traveling Salesman Problem TSP $(G(V, E), c, min)$.

Basic idea: 3 arcs are removed at each step, then the 3 paths are reconnected with 3 different arcs.

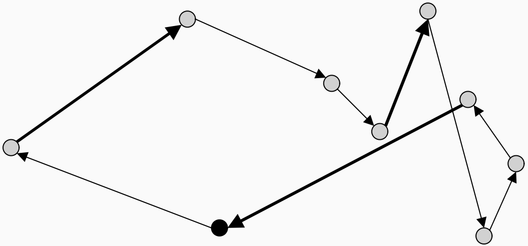
Effectiveness: 4% more than the minimum, on average.

The idea can be generalized up until the k-opt. For $k = n$ the neighborhood becomes exact and contains all the possible hamiltonian cycles, which are $n!$.

Initial cycle:



New cycle:



Meta-heuristic techniques

Problem:

Heuristic descent methods risk getting stuck in local optima that are not global optima.

Solution: meta-heuristic algorithms:

- local search algorithms that emply special techniques to escape local minima;
- algorithms that have to avoid cycles (repetition of already visited solutions).

Examples:

- **Simulated Annealing (SA)**
- **Tabu Search (TS):** allows a move towards a solution worse than the current one and avoids to visit already seen solutions by means of a list of forbidden (**tabu**) moves.
- **Genetic Algorithms (GA)**