Constructive heuristics

Celso C. Ribeiro (celso@ic.uff.br)

University of Vienna

Metaheuristics – 2017-10-18

Overview of talk

- Solution construction
  - Greedy algorithms
  - Adaptive greedy algorithms
  - Semi-greedy algorithms
  - Random multistart
  - Semi-greedy multistart
  - Semi-greedy construction
- Concluding remarks

Solution construction – Greedy algorithms

- Feasible solution S of a combinatorial optimization problem is subset of ground set $E = \{1, \ldots, n\}$.
- Since certain subsets of ground set elements cause infeasibilities, then a feasible solution cannot contain any such subset.

Solution construction – Greedy algorithms

- Feasible solution S of a combinatorial optimization problem is subset of ground set $E = \{1, \ldots, n\}$.
- Since certain subsets of ground set elements cause infeasibilities, then a feasible solution cannot contain any such subset.
- If c_i is the contribution of ground set element $i \in E$ to the objective function, we assume that $f(S) = \sum_{i \in S} c_i$.
- We build a solution incrementally from scratch.
  - At each step, a single ground set element is added to the partial solution under construction.

Solution construction – Greedy algorithms

- Feasible solution S of a combinatorial optimization problem is subset of ground set $E = \{1, \ldots, n\}$.
- Since certain subsets of ground set elements cause infeasibilities, then a feasible solution cannot contain any such subset.
- If c_i is the contribution of ground set element $i \in E$ to the objective function, we assume that $f(S) = \sum_{i \in S} c_i$.
- We build a solution incrementally from scratch.
  - At each step, a single ground set element is added to the partial solution under construction.
  - A ground set element to be added at each step cannot be such that its combination with one or more previously added elements leads to an infeasibility.
  - We call such an element *feasible* and denote by $\mathcal{F}$ the set of all feasible elements at the time a given step is performed.

Solution construction – Greedy algorithms

- Feasible solution S of a combinatorial optimization problem is subset of ground set $E = \{1, \dots, n\}$.
- Since certain subsets of ground set elements cause infeasibilities, then a feasible solution cannot contain any such subset.
- If c_i is the contribution of ground set element $i \in E$ to the objective function, we assume that $f(S) = \sum_{i \in S} c_i$.
- We build a solution incrementally from scratch.
  - At each step, a single ground set element is added to the partial solution under construction.
  - We call such an element *feasible* and denote by $\mathcal{F}$ the set of all feasible elements at the time a given step is performed.

Since the set of candidate elements $\mathcal{F}$ may contain more than one element, an algorithm designed to build a feasible solution for some problem must have a mechanism to select the next feasible ground set element from $\mathcal{F}$ to be added to the partially built solution under construction.

- From among all yet unselected feasible elements, a *greedy algorithm* chooses one of least cost.

Solution construction – Greedy algorithms

- The pseudo-code shows a greedy algorithm for a minimization problem.

begin GREEDY;
$S \leftarrow \varnothing$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
while $\mathcal{F} \neq \varnothing$ **do**
    $i^* \leftarrow \text{argmin}\{c_i \ : \ i \in \mathcal{F}\}$;
    $S \leftarrow S \cup \{i^*\}$;
    $f(S) \leftarrow f(S) + c_{i^*}$;
    $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
end-while;
return $S, f(S)$;
end GREEDY.

Solution construction – Greedy algorithms

- The pseudo-code shows a greedy algorithm for a minimization problem.
- Feasible solution S is constructed, one ground set element at a time.
- $\mathcal{F}$ is set of feasible ground set elements.
- Greedy algorithm selects feasible ground set element of smallest cost.
- Note that costs can be sorted in a preprocessing step.

begin GREEDY;
$S \leftarrow \varnothing$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\} \text{ is not infeasible}\}$;
while $\mathcal{F} \neq \varnothing$ **do**
    $i^* \leftarrow \text{argmin}\{c_i \ : \ i \in \mathcal{F}\}$;
    $S \leftarrow S \cup \{i^*\}$;
    $f(S) \leftarrow f(S) + c_{i^*}$;
    $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\} \text{ is not infeasible}\}$;
end-while;
return $S, f(S)$;
end GREEDY.

Solution construction – Greedy algorithms

- The pseudo-code shows a greedy algorithm for a minimization problem.
- Feasible solution S is constructed, one ground set element at a time.
- $\mathcal{F}$ is set of feasible ground set elements.
- Greedy algorithm selects feasible ground set element of smallest cost.
- Note that costs can be sorted in a preprocessing step.
- Example: Greedy algorithm for minimum weight spanning tree (Kruskal, 1957).

begin GREEDY;
$S \leftarrow \emptyset$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
while $\mathcal{F} \neq \emptyset$ **do**
    $i^* \leftarrow \text{argmin}\{c_i \ : \ i \in \mathcal{F}\}$;
    $S \leftarrow S \cup \{i^*\}$;
    $f(S) \leftarrow f(S) + c_{i^*}$;
    $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
end-while;
return $S, f(S)$;
end GREEDY.

Solution construction – Adaptive greedy algorithms

- The greedy algorithm in the previous slide selects an element i^* of the set of feasible candidate elements $\mathcal{F}$ as $i^* \leftarrow \text{argmin}\{c_i \ : \ i \in \mathcal{F}\}$, where c_i is the cost associated with the inclusion of element $i \in \mathcal{F}$ in the solution.
- In that algorithm, only this constant cost is used to guide the algorithm, and therefore the elements can be sorted in the increasing order of their costs in a preprocessing step.

Solution construction – Adaptive greedy algorithms

- The greedy algorithm in the previous slide selects an element i^* of the set of feasible candidate elements $\mathcal{F}$ as $i^* \leftarrow \text{argmin}\{c_i \ : \ i \in \mathcal{F}\}$, where c_i is the cost associated with the inclusion of element $i \in \mathcal{F}$ in the solution.
- In that algorithm, only this constant cost is used to guide the algorithm, and therefore the elements can be sorted in the increasing order of their costs in a preprocessing step.
- Although that greedy algorithm is applicable in many situations, such as to the minimum spanning tree problem, there are other situations where a different measure of the contribution of an element guides the algorithm and it is affected by the previous choices of elements made by the algorithm.
- We call these adaptive greedy algorithms.

Solution construction – Adaptive greedy algorithms

- The pseudo-code shows a generic adaptive greedy algorithm for a minimization problem.

begin ADAPTIVE-GREEDY;
$S \leftarrow \varnothing$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
Compute the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
while $\mathcal{F} \neq \varnothing$ **do**
    $i^* \leftarrow \text{argmin}\{g(i) \ : \ i \in \mathcal{F}\}$;
    $S \leftarrow S \cup \{i^*\}$;
    $f(S) \leftarrow f(S) + c_{i^*}$;
    $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
    Update the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
end-while;
return $S, f(S)$;
end ADAPTIVE-GREEDY.

Solution construction – Adaptive greedy algorithms

- The pseudo-code shows a generic adaptive greedy algorithm for a minimization problem.
- Feasible solution S is constructed, one ground set element at a time.
- $\mathcal{F}$ is set of feasible ground set elements.
- Greedy choice function $g(i)$ is the "contribution" of ground set element $i \in \mathcal{F}$.
- Adaptive greedy algorithm selects feasible ground set element of smallest greedy choice function.

begin ADAPTIVE-GREEDY;
    $S \leftarrow \varnothing$;
    $f(S) \leftarrow 0$;
    $\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
    Compute the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
    while $\mathcal{F} \neq \varnothing$ **do**
        $i^* \leftarrow \text{argmin}\{g(i) \ : \ i \in \mathcal{F}\}$;
        $S \leftarrow S \cup \{i^*\}$;
        $f(S) \leftarrow f(S) + c_{i^*}$;
        $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
        Update the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
    end-while;
    return $S, f(S)$;
end ADAPTIVE-GREEDY.

Solution construction – Adaptive greedy algorithms

- The pseudo-code shows a generic adaptive greedy algorithm for a minimization problem.
- Feasible solution S is constructed, one ground set element at a time.
- $\mathcal{F}$ is set of feasible ground set elements.
- Greedy choice function $g(i)$ is the "contribution" of ground set element $i \in \mathcal{F}$.
- Adaptive greedy algorithm selects feasible ground set element of smallest greedy choice function.
- Example: Adaptive greedy algorithm for set covering (Johnson, 1974).

begin ADAPTIVE-GREEDY;
  $S \leftarrow \varnothing$;
  $f(S) \leftarrow 0$;
  $\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
  Compute the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
  while $\mathcal{F} \neq \varnothing$ **do**
    $i^* \leftarrow \text{argmin}\{g(i) \ : \ i \in \mathcal{F}\}$;
    $S \leftarrow S \cup \{i^*\}$;
    $f(S) \leftarrow f(S) + c_{i^*}$;
    $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
    Update the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
  end-while;
  return $S, f(S)$;
end ADAPTIVE-GREEDY.

TSP – Adaptive greedy algorithm

- The algorithm on the right is a nearest neighbor adaptive greedy algorithm for the TSP.

begin ADAPTIVE-GREEDY-TSP;

$S \leftarrow \emptyset$;

$f(S) \leftarrow 0$;

Let i be any node in V and set $i_0 \leftarrow i$;

$\mathcal{F} \leftarrow V \setminus \{i_0\}$;

while $\mathcal{F} \neq \emptyset$ **do**

  $\mathcal{H} \leftarrow \{j \in \mathcal{F} \ : \ (i,j) \in U\}$;

  $g(j) \leftarrow d_{ij}$ for all $j \in \mathcal{H}$;

  $j' \leftarrow \text{argmin}\{g(j) \ : \ j \in \mathcal{H}\}$;

  $S \leftarrow S \cup \{(i,j')\}$;

  $f(S) \leftarrow f(S) + d_{i,j'}$;

  $\mathcal{F} \leftarrow \mathcal{F} \setminus \{j'\}$;

  $i \leftarrow j'$;

end-while;

$S \leftarrow S \cup \{(i, i_0)\}$;

$f(S) \leftarrow f(S) + d_{i,i_0}$;

return $S, f(S)$;

end ADAPTIVE-GREEDY-TSP.

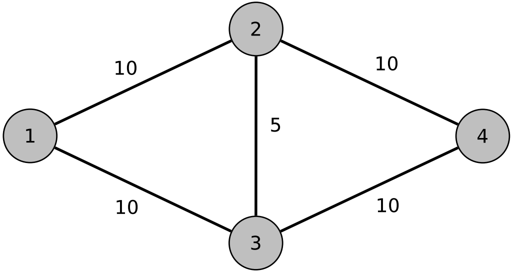
TSP – Adaptive greedy algorithm

- The algorithm on the right is a nearest neighbor adaptive greedy algorithm for the TSP.
- Given a graph $G = (V, U)$, where V is the set of nodes and U is the set of weighted edges, let d_{ij} be the length (or weight) of edge $(i, j) \in U$.
- An adaptive greedy approach for this problem is to grow the set of visited nodes of the tour, starting from any initial node i_0.
- Denote by v the last visited node of the partial tour under construction. At each step we use the greedy choice function to select a nearest unvisited node adjacent to v. This node is added to the tour.
- This is repeated until the tour visits all nodes.

begin ADAPTIVE-GREEDY-TSP;
$S \leftarrow \varnothing$;
$f(S) \leftarrow 0$;
Let i be any node in V and set $i_0 \leftarrow i$;
$\mathcal{F} \leftarrow V \setminus \{i_0\}$;
while $\mathcal{F} \neq \varnothing$ **do**
    $\mathcal{H} \leftarrow \{j \in \mathcal{F} \ : \ (i, j) \in U\}$;
    $g(j) \leftarrow d_{ij}$ for all $j \in \mathcal{H}$;
    $j' \leftarrow \text{argmin}\{g(j) \ : \ j \in \mathcal{H}\}$;
    $S \leftarrow S \cup \{(i, j')\}$;
    $f(S) \leftarrow f(S) + d_{i,j'}$;
    $\mathcal{F} \leftarrow \mathcal{F} \setminus \{j'\}$;
    $i \leftarrow j'$;
end-while;
$S \leftarrow S \cup \{(i, i_0)\}$;
$f(S) \leftarrow f(S) + d_{i,i_0}$;
return $S, f(S)$;
end ADAPTIVE-GREEDY-TSP.

Solution construction – Semi-greedy algorithms

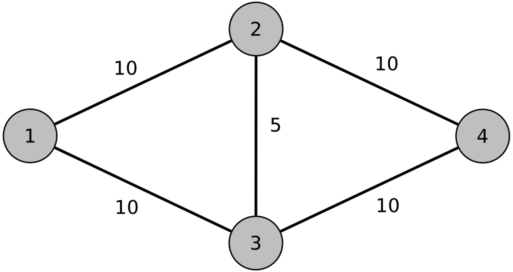
Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.



Solution construction – Semi-greedy algorithms

Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

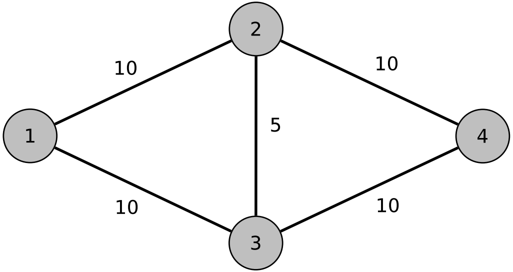
- The algorithm starts from any node and repeatedly moves from the current node to its nearest unvisited node.



Solution construction – Semi-greedy algorithms

Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

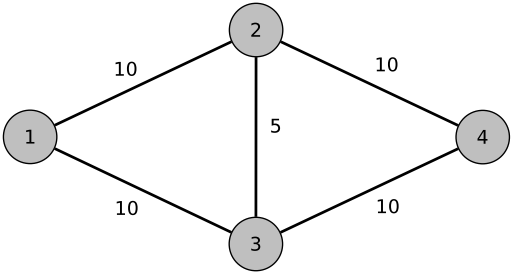
- The algorithm starts from any node and repeatedly moves from the current node to its nearest unvisited node.
- Suppose the algorithm were to start from node 1, in which case it should move next to either node 2 or 3.



Solution construction – Semi-greedy algorithms

Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

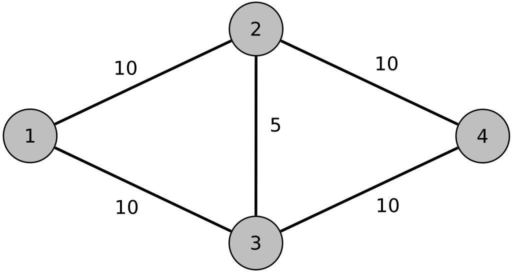
- The algorithm starts from any node and repeatedly moves from the current node to its nearest unvisited node.
- Suppose the algorithm were to start from node 1, in which case it should move next to either node 2 or 3.
- If it moves to node 2, then it must necessarily move next to node 3 and then to node 4. Since there is no edge connecting node 4 to node 1, the algorithm will **fail to find a tour**.



Solution construction – Semi-greedy algorithms

Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

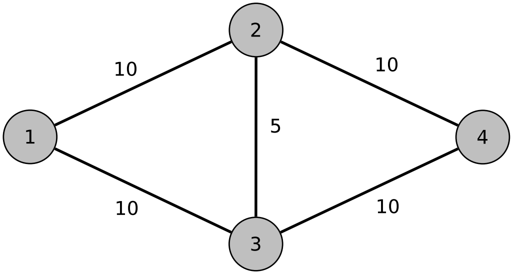
- By symmetry, the same situation occurs if it were to start from node 4.



Solution construction – Semi-greedy algorithms

Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

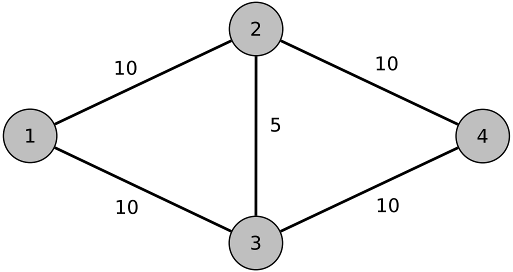
- By symmetry, the same situation occurs if it were to start from node 4.
- Now suppose the algorithm starts from node 2. Node 3 is the nearest to node 2 and from node 3 it can move either to node 1 or node 4, **failing in either case to find a tour**.
- Again, by symmetry, the same situation occurs if one were to start from node 3.



Solution construction – Semi-greedy algorithms

Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

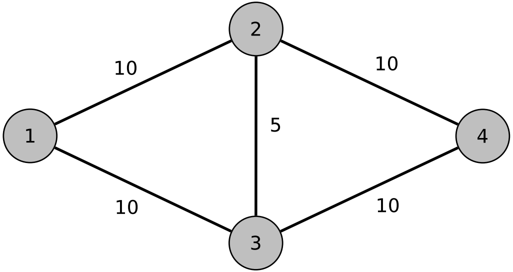
- By symmetry, the same situation occurs if it were to start from node 4.
- Now suppose the algorithm starts from node 2. Node 3 is the nearest to node 2 and from node 3 it can move either to node 1 or node 4, **failing in either case to find a tour**.
- Again, by symmetry, the same situation occurs if one were to start from node 3.
- Therefore, this adaptive greedy **algorithm fails to find a tour**, no matter which node it starts from.



Solution construction – Semi-greedy algorithms

Consider the following **randomized version** of the same adaptive greedy algorithm: Start from any node and repeatedly move, with equal probability, to one of its two nearest unvisited nodes.

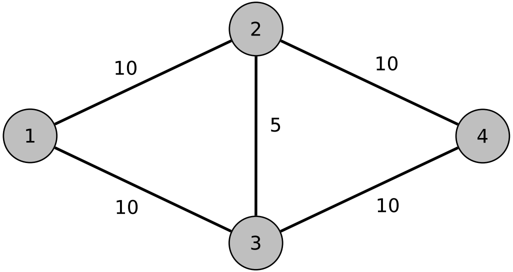
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.



Solution construction – Semi-greedy algorithms

Consider the following **randomized version** of the same adaptive greedy algorithm: Start from any node and repeatedly move, with equal probability, to one of its two nearest unvisited nodes.

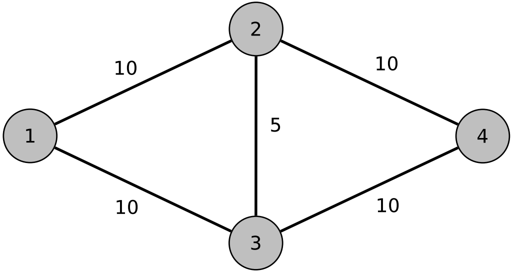
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.
  - On the one hand, if it were to move to node 3, it would fail to find a tour.
  - On the other hand, by moving to node 4, it would then go to node 3, and then back to node 1, thus finding a tour of length 40.



Solution construction – Semi-greedy algorithms

Consider the following **randomized version** of the same adaptive greedy algorithm: Start from any node and repeatedly move, with equal probability, to one of its two nearest unvisited nodes.

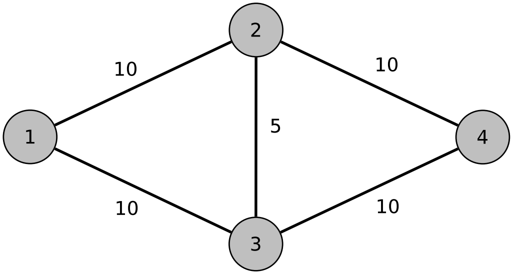
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.
  - On the one hand, if it were to move to node 3, it would fail to find a tour.
  - On the other hand, by moving to node 4, it would then go to node 3, and then back to node 1, thus finding a tour of length 40.
- Therefore, there is a 50% probability that the algorithm will find a tour if it starts from node 1.



Solution construction – Semi-greedy algorithms

Consider the following **randomized version** of the same adaptive greedy algorithm: Start from any node and repeatedly move, with equal probability, to one of its two nearest unvisited nodes.

- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.
  - On the one hand, if it were to move to node 3, it would fail to find a tour.
  - On the other hand, by moving to node 4, it would then go to node 3, and then back to node 1, thus finding a tour of length 40.
- Therefore, there is a 50% probability that the algorithm will find a tour if it starts from node 1.
- With repeated applications, the probability of finding the optimal cycle quickly approaches one.



After ten attempts, the probability of finding the optimal solution is over 99.9%.

Semi-greedy algorithms

Algorithms like the one in the previous slide, which add randomization to a greedy or adaptive greedy algorithm, are called semi-greedy or randomized-greedy algorithms.

- The pseudo-code on the right shows a semi-greedy algorithm for a minimization problem.

```
begin SEMI-GREEDY;
  S ← ∅;
  f(S) ← 0;
  𝓕 ← {i ∈ E : S ∪ {i} is not infeasible};
  while 𝓕 ≠ ∅ do
      Let RCL be a subset of low-cost elements of 𝓕;
      Let i* be a randomly chosen element from RCL;
      S ← S ∪ {i*};
      f(S) ← f(S) + c_{i*};
      𝓕 ← {i ∈ 𝓕 \ {i*} : S ∪ {i} is not infeasible};
  end-while;
  return S, f(S);
end SEMI-GREEDY.
```

Semi-greedy algorithms

Algorithms like the one in the previous slide, which add randomization to a greedy or adaptive greedy algorithm, are called semi-greedy or randomized-greedy algorithms.

- The pseudo-code on the right shows a semi-greedy algorithm for a minimization problem.
- It is similar to a greedy algorithm, differing only in how the ground set element is chosen from the set $\mathcal{F}$ of feasible candidate ground set elements (lines 5 and 6).
- In line 5, a subset of low-cost elements of set $\mathcal{F}$ is placed in a restricted candidate list (RCL).
- In line 6, a ground set element is selected at random from the RCL to be incorporated into the solution in line 7.

begin SEMI-GREEDY;
1 $S \leftarrow \varnothing$;
2 $f(S) \leftarrow 0$;
3 $\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
4 **while** $\mathcal{F} \neq \varnothing$ **do**
5   Let RCL be a subset of low-cost elements of $\mathcal{F}$;
6   Let i^* be a randomly chosen element from RCL;
7     $S \leftarrow S \cup \{i^*\}$;
8     $f(S) \leftarrow f(S) + c_{i^*}$;
9     $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
10 **end-while**;
11 **return** $S, f(S)$;
end SEMI-GREEDY.

Semi-greedy algorithms: Building the RCL

Two simple schemes to define a restricted candidate list are:

- Cardinality-based RCL: The k least-costly feasible candidate ground set elements of set $\mathcal{F}$ are placed in the RCL.

Semi-greedy algorithms: Building the RCL

Two simple schemes to define a restricted candidate list are:

- Cardinality-based RCL: The k least-costly feasible candidate ground set elements of set $\mathcal{F}$ are placed in the RCL.
- Quality-based RCL: RCL is formed by all ground-set elements $i \in \mathcal{F}$ satisfying

$$c_{\min} \leq c_i \leq c_{\min} + \alpha(c_{\max} - c_{\min}),$$

  where

$$c_{\min} = \min\{c_i \;:\; i \in \mathcal{F}\},\ c_{\max} = \max\{c_i \;:\; i \in \mathcal{F}\}, \text{ and } 0 \leq \alpha \leq 1.$$

  Note that setting
  - $\alpha = 0$ corresponds to a pure greedy algorithm, since a lowest cost element will always be selected.
  - $\alpha = 1$ leads to a random algorithm, since any new element may be added with equal probability.

Random multi-start

A multistart procedure is an algorithm which repeatedly applies a solution construction procedure and outputs the best solution found over all trials. Each trial, or iteration, of a multistart procedure is applied under different conditions.

- The pseudo-code on the right is of a random multistart procedure for a minimization problem.

```
begin RANDOM-MULTISTART;
  f* ← ∞;
  while stopping criterion not satisfied do
      S ← RandomSolution;
      if f(S) < f* then
          S* ← S;
          f* ← f(S);
      end-if;
  end-while;
  return S*;
end RANDOM-MULTISTART.
```

Random multi-start

A multistart procedure is an algorithm which repeatedly applies a solution construction procedure and outputs the best solution found over all trials. Each trial, or iteration, of a multistart procedure is applied under different conditions.

- The pseudo-code on the right is of a random multistart procedure for a minimization problem.
- Like the GREEDY algorithm, a new random solution is generated in line 3 by adding to the partial solution (initially empty) a new feasible ground set element, one element at a time.
- Unlike GREEDY, each ground set element is chosen at random from the set of candidate ground set elements.

```
begin RANDOM-MULTISTART;
1  f* ← ∞;
2  while stopping criterion not satisfied do
3      S ← RandomSolution;
4      if f(S) < f* then
5          S* ← S;
6          f* ← f(S);
7      end-if;
8  end-while;
9  return S*;
end RANDOM-MULTISTART.
```

Semi-greedy multi-start

The semi-greedy algorithm can be embedded in a multistart framework.

- The pseudo-code on the right is of a semi-greedy multistart procedure for a minimization problem.

```
begin SEMI-GREEDY-MULTISTART;
f* ← ∞;
while stopping criterion not satisfied do
    S ← SEMI-GREEDY;
    if f(S) < f* then
        S* ← S;
        f* ← f(S);
    end-if;
end-while;
return S*;
end SEMI-GREEDY-MULTISTART.
```

Semi-greedy multi-start

The semi-greedy algorithm can be embedded in a multistart framework.

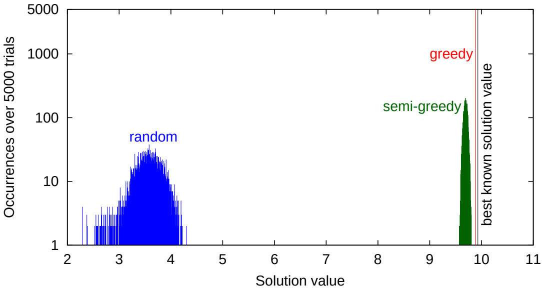
- The pseudo-code on the right is of a semi-greedy multistart procedure for a minimization problem.
- This algorithm is almost identical to the random multistart method, except that solutions are generated with a semi-greedy procedure instead of at random.
- Note that each invocation of the semi-greedy procedure in line 3 is independent of the others, therefore producing independent solutions.

```
begin SEMI-GREEDY-MULTISTART;
1  f* ← ∞;
2  while stopping criterion not satisfied do
3       S ← SEMI-GREEDY;
4       if f(S) < f* then
5            S* ← S;
6            f* ← f(S);
7       end-if;
8  end-while;
9  return S*;
end SEMI-GREEDY-MULTISTART.
```

Semi-greedy multistart

Recall that parameter α in a semi-greedy construction procedure controls the mix of greediness and randomness in the constructed solution.

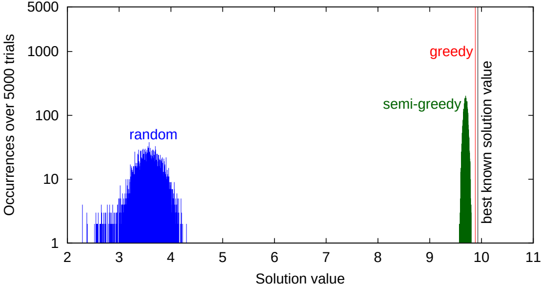
- In the case of a maximization problem:
  - $\alpha = 1$ leads to a greedy construction.
  - $\alpha = 0$ leads to a random construction.



Semi-greedy multistart

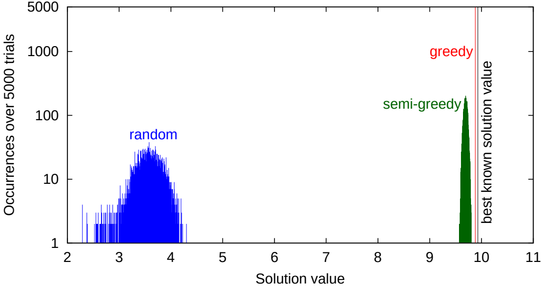
Recall that parameter α in a semi-greedy construction procedure controls the mix of greediness and randomness in the constructed solution.

- In the case of a maximization problem:
  - $\alpha = 1$ leads to a greedy construction.
  - $\alpha = 0$ leads to a random construction.
- The figure shows the distribution of solution values on an instance of the *maximum covering problem* produced by
  - a random multistart procedure,
  - a semi-greedy multistart algorithm with the RCL parameter $\alpha = 0.85$,
  - a greedy algorithm,
  - along with the best known solution value.



Semi-greedy multistart

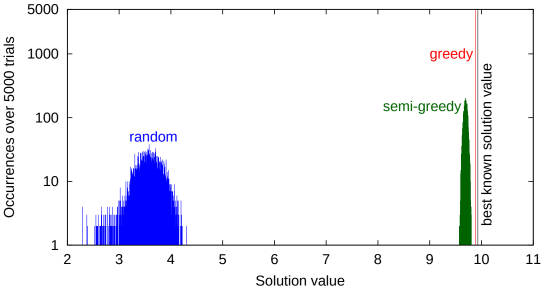
The figure compares the two distributions with the greedy solution value and the **best-known solution** value for this maximization problem. It illustrates four important points:



Semi-greedy multistart

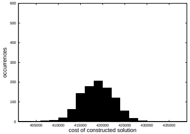
The figure compares the two distributions with the greedy solution value and the **best-known solution** value for this maximization problem. It illustrates four important points:

1. Semi-greedy solutions are on average much better than random solutions.
2. There is more variance in the solution values produced by a random multistart method than by a semi-greedy multistart algorithm.
3. The greedy solution is on average better than both the random and the semi-greedy solutions but, even if ties are broken at random, it has less variance than the random or semi-greedy solutions.
4. Random, semi-greedy, and greedy solutions are usually sub-optimal.

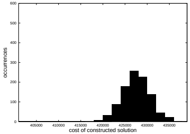


Semi-greedy algorithm

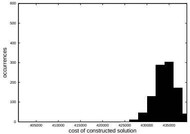
Distribution of semi-greedy solution values as a function of the quality-based RCL parameter α (1000 repetitions were recorded for each value of α) on an instance of the maximum weighted satisfiability problem.



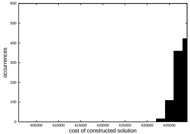
$\alpha = 0$ (random)



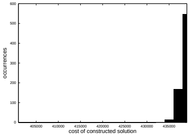
$\alpha = 0.2$



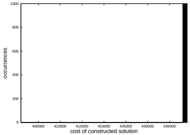
$\alpha = 0.4$



$\alpha = 0.6$



$\alpha = 0.8$

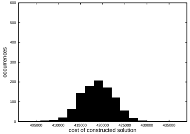


$\alpha = 1$ (greedy)

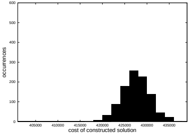
Semi-greedy algorithm

Distribution of semi-greedy solution values as a function of the quality-based RCL parameter α (1000 repetitions were recorded for each value of α) on an instance of the maximum weighted satisfiability problem.

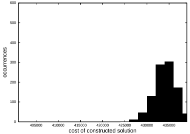
- As α increases from 0 (random construction) to 1 (greedy construction):



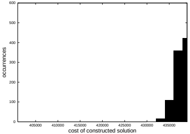
$\alpha = 0$ (random)



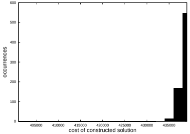
$\alpha = 0.2$



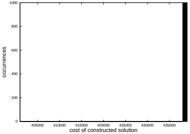
$\alpha = 0.4$



$\alpha = 0.6$



$\alpha = 0.8$

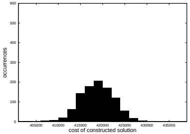


$\alpha = 1$ (greedy)

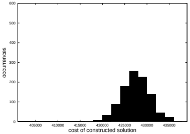
Semi-greedy algorithm

Distribution of semi-greedy solution values as a function of the quality-based RCL parameter α (1000 repetitions were recorded for each value of α) on an instance of the maximum weighted satisfiability problem.

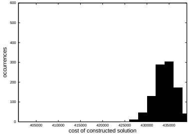
- As α increases from 0 (random construction) to 1 (greedy construction):
  - Average solution value increases.



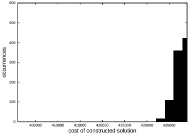
$\alpha = 0$ (random)



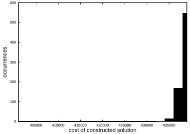
$\alpha = 0.2$



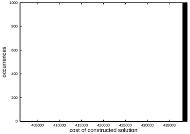
$\alpha = 0.4$



$\alpha = 0.6$



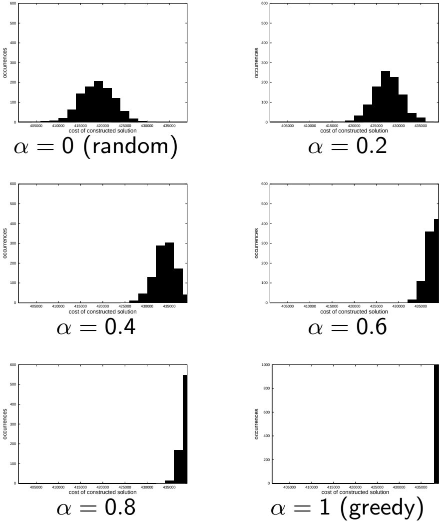
$\alpha = 0.8$



$\alpha = 1$ (greedy)

Distribution of semi-greedy solution values as a function of the quality-based RCL parameter α (1000 repetitions were recorded for each value of α) on an instance of the maximum weighted satisfiability problem.

- As α increases from 0 (random construction) to 1 (greedy construction):
  - Average solution value increases.
  - Spread of solution values decreases.



Concluding remarks

The material in this talk is taken from

- Chapter 3 – Solution construction and greedy algorithms

of our book, *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures* (Resende & Ribeiro, Springer, 2016).