GRASP – Greedy Randomized Adaptive Search Procedures

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

University of Vienna

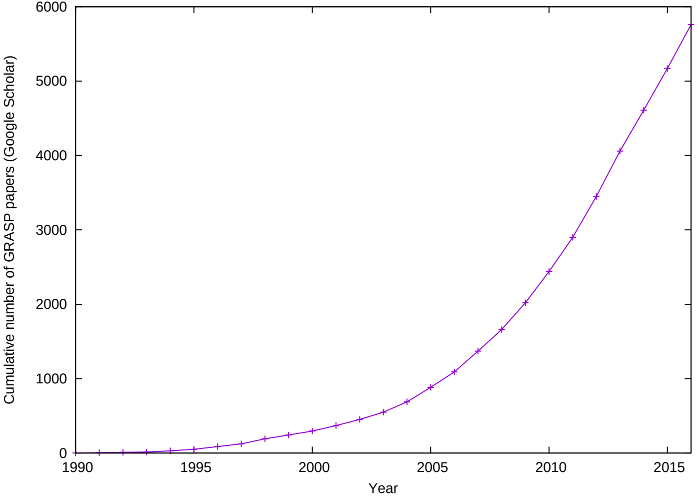
Metaheuristics – 2017-11-15

Overview of talk

- Solution construction
  - Greedy algorithms
  - Adaptive greedy algorithms
  - Semi-greedy algorithms
  - Random multistart
  - Semi-greedy multistart
- The basic GRASP heuristic
  - Random multistart
  - Semi-greedy multistart
  - GRASP
- Example of a GRASP for maximum cut in a graph
  - Problem definition
  - Semi-greedy construction
  - Local search
- Concluding remarks

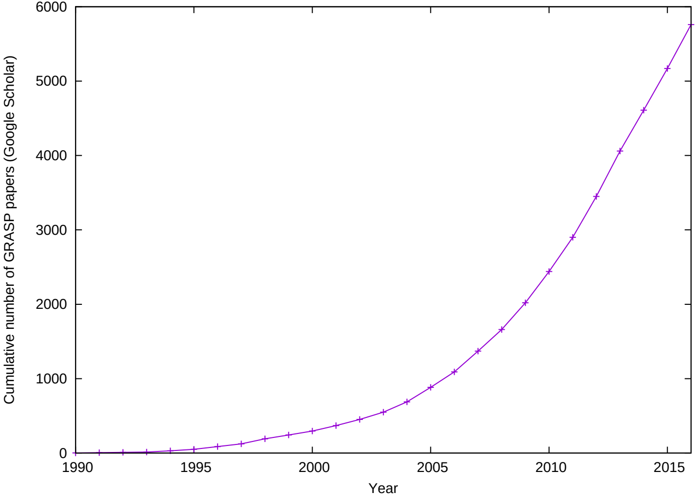
GRASP papers over time

- GRASP was introduced in by Feo & Resende in 1989.



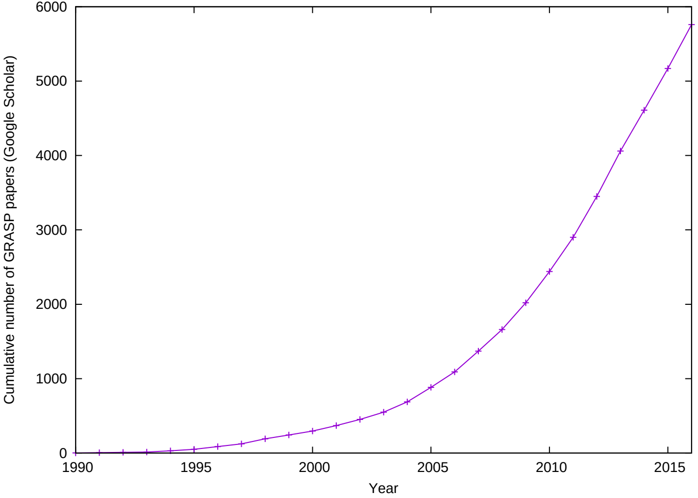
GRASP papers over time

- GRASP was introduced in by Feo & Resende in 1989.
- Book by Resende & Ribeiro appeared in 2016.



GRASP papers over time

- GRASP was introduced in by Feo & Resende in 1989.
- Book by Resende & Ribeiro appeared in 2016.
- Number of papers on GRASP continues to grow. In 2016, almost 600 papers were published.



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.
- The 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 a minimum cost element i^* of the set of feasible candidate elements to incorporate 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 a minimum cost element i^* of the set of feasible candidate elements to incorporate 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.
- Feasible solution S is constructed, one ground set element at a time.
- $\mathcal{F}$ is set of feasible ground set elements.

begin ADAPTIVE-GREEDY;
$S \leftarrow \emptyset$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
Compute the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
while $\mathcal{F} \neq \emptyset$ **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;
$S \leftarrow \emptyset$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
Compute the greedy choice function $g(i)$ for all $i \in \mathcal{F}$;
while $\mathcal{F} \neq \emptyset$ **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.

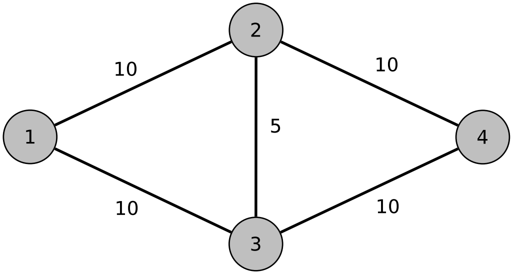
Solution construction – Adaptive greedy algorithms

- 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 nearest neighbor heuristic for TSP.

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 – Semi-greedy algorithms

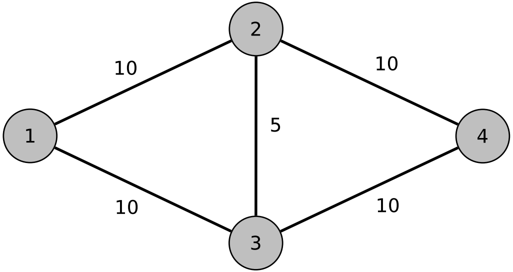
Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.



Solution construction – Semi-greedy algorithms

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

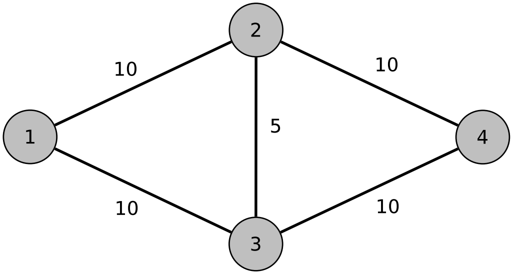
- The algorithm starts from any node and repeatedly moves from the current node to its nearest unvisited node.



Solution construction – Semi-greedy algorithms

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

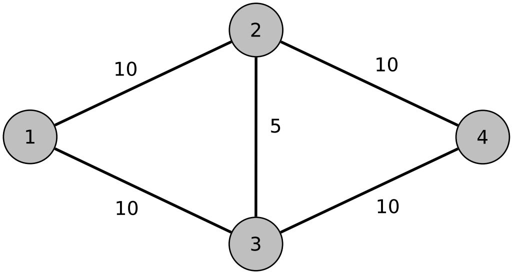
- The algorithm starts from any node and repeatedly moves from the current node to its nearest unvisited node.
- Suppose the algorithm were to start from node 1, in which case it should move next to either node 2 or 3.



Solution construction – Semi-greedy algorithms

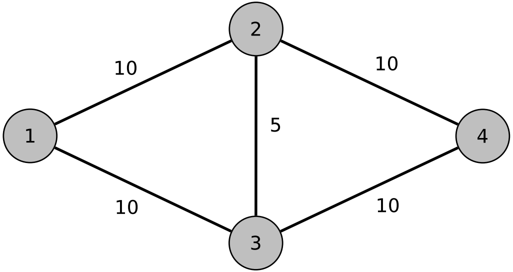
Suppose we wish to find a shortest Hamiltonian cycle in this graph applying the nearest neighbor adaptive greedy algorithm.

- 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**.
- By symmetry reasoning, we show this adaptive greedy **algorithm fails to find a tour**, no matter which node it starts from.



Solution construction – Semi-greedy algorithms

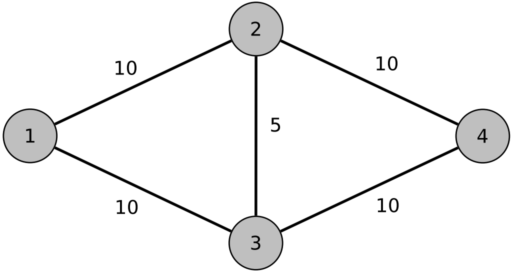
Consider the following **randomized version** of the same adaptive greedy algorithm: Start from any node and repeatedly move, with equal probability, to one of its two nearest unvisited nodes.



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.

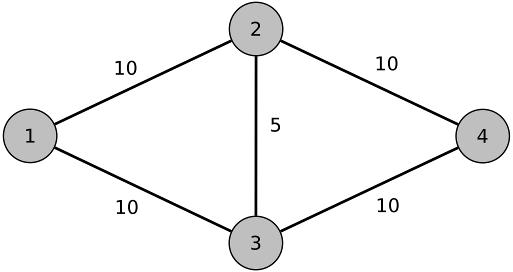
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.



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.

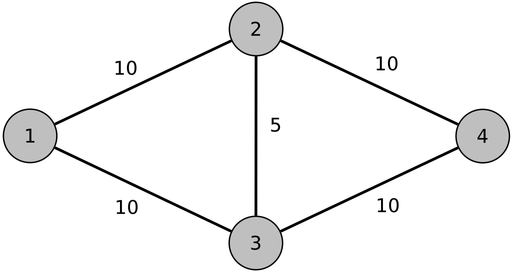
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.



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.

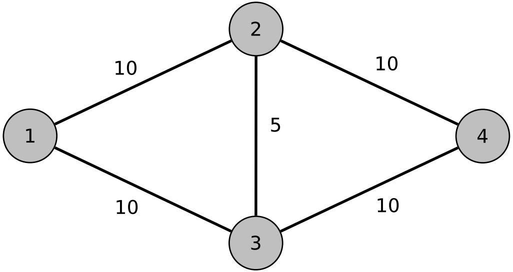
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.
  - On the one hand, if it were to move to node 3, it would fail to find a tour.



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.

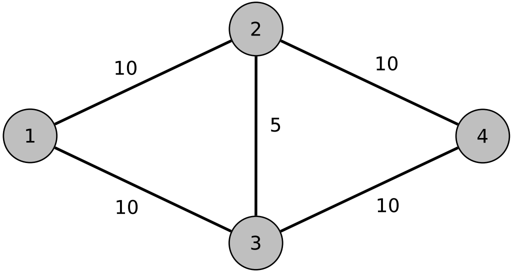
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.
  - On the one hand, if it were to move to node 3, it would fail to find a tour.
  - On the other hand, 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.

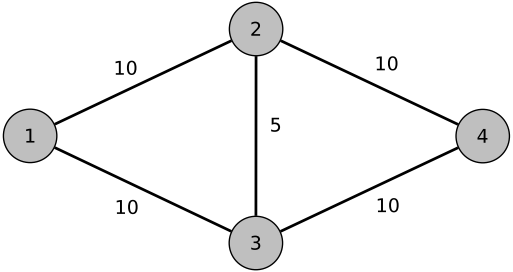
- Starting from node 1, it then moves to either node 2 or node 3 with equal probability.
- Suppose it were to move to node 2. Now, again with equal probability, it moves to either node 3 or node 4.
  - On the one hand, if it were to move to node 3, it would fail to find a tour.
  - On the other hand, 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;
$S \leftarrow \varnothing$;
$f(S) \leftarrow 0$;
$\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
while $\mathcal{F} \neq \varnothing$ **do**
    Let RCL be a subset of low-cost elements of $\mathcal{F}$;
    Let i^* be a randomly chosen element from RCL;
    $S \leftarrow S \cup \{i^*\}$;
    $f(S) \leftarrow f(S) + c_{i^*}$;
    $\mathcal{F} \leftarrow \{i \in \mathcal{F} \setminus \{i^*\} \ : \ S \cup \{i\}$ is not infeasible$\}$;
end-while;
return $S, f(S)$;
end 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 $S \leftarrow \emptyset$;
2 $f(S) \leftarrow 0$;
3 $\mathcal{F} \leftarrow \{i \in E \ : \ S \cup \{i\}$ is not infeasible$\}$;
4 **while** $\mathcal{F} \neq \emptyset$ **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:

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

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

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.

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;
  $f^* \leftarrow \infty$;
  while stopping criterion not satisfied **do**
    $S \leftarrow$ `RandomSolution`;
    if $f(S) < f^*$ **then**
      $S^* \leftarrow S$;
      $f^* \leftarrow f(S)$;
    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 $f^* \leftarrow \infty$;
2 **while** stopping criterion not satisfied **do**
3 $\quad S \leftarrow$ `RandomSolution`;
4 $\quad$ **if** $f(S) < f^*$ **then**
5 $\quad\quad S^* \leftarrow S$;
6 $\quad\quad f^* \leftarrow f(S)$;
7 $\quad$ **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.

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

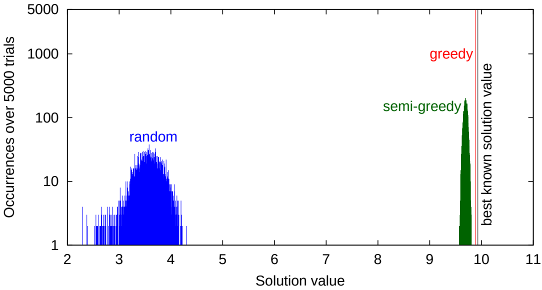
The semi-greedy algorithm can be embedded in a multistart framework.

- 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

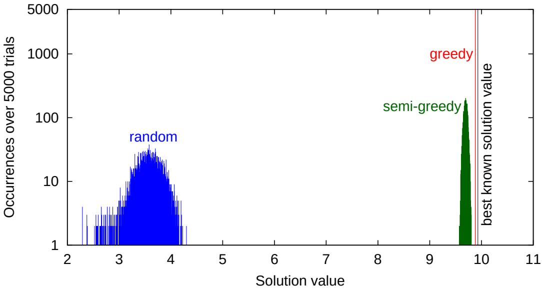
Recall that parameter α in a semi-greedy construction procedure controls the mix of greediness and randomness in the constructed solution.



Semi-greedy multistart

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

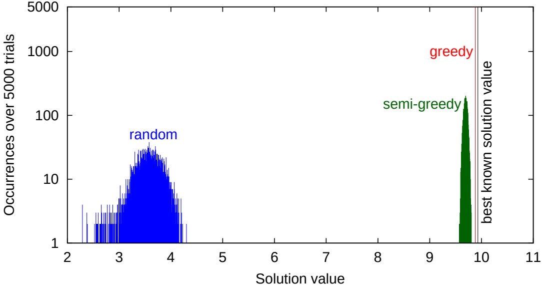
- In the case of a maximization problem:
  - $\alpha = 1$ leads to a greedy construction.
  - $\alpha = 0$ leads to a random construction.



Semi-greedy multistart

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

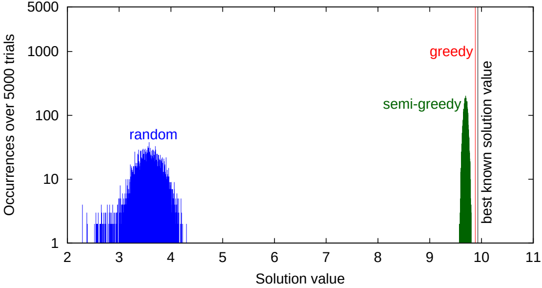
- In the case of a maximization problem:
  - $\alpha = 1$ leads to a greedy construction.
  - $\alpha = 0$ leads to a random construction.
- The figure shows the distribution of solution values on an instance of the *maximum covering problem* produced by



Semi-greedy multistart

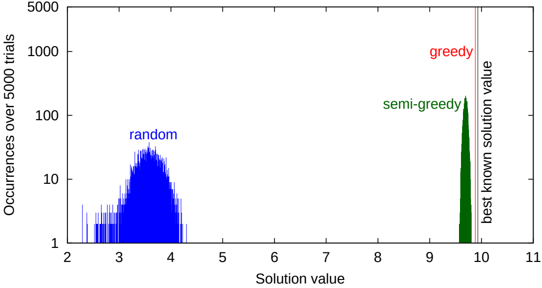
Recall that parameter α in a semi-greedy construction procedure controls the mix of greediness and randomness in the constructed solution.

- 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

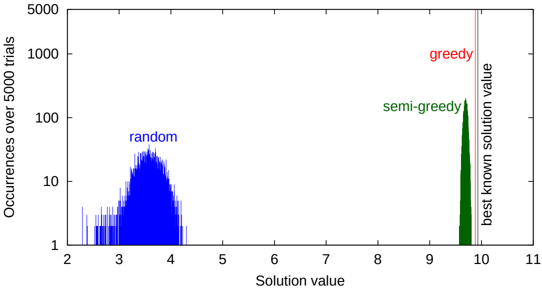
The figure compares the two distributions with the greedy solution value and the **best-known solution** value for this maximization problem. It illustrates four important points:



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:

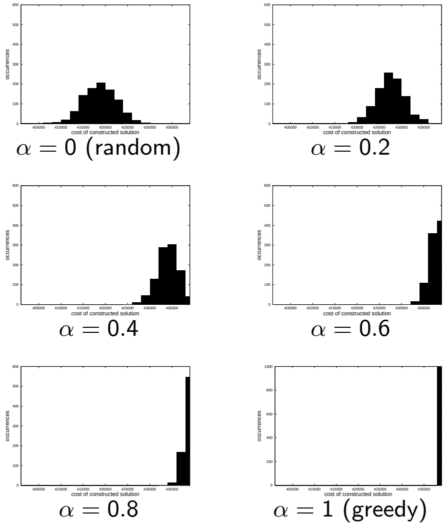
1. Semi-greedy solutions are on average much better than random solutions.
2. There is more variance in the solution values produced by a random multistart method than by a semi-greedy multistart algorithm.
3. The greedy solution is on average better than both the random and the semi-greedy solutions 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

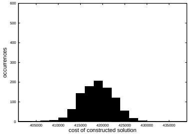
Distribution of semi-greedy solution values as a function of the quality-based RCL parameter α (1000 repetitions were recorded for each value of α) on an instance of the maximum weighted satisfiability problem.

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

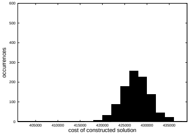


Distribution of semi-greedy solution values as a function of the quality-based RCL parameter α (1000 repetitions were recorded for each value of α) on an instance of the maximum weighted satisfiability problem.

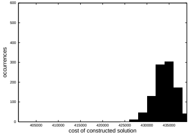
- As α increases from 0 (random construction) to 1 (greedy construction):
  - Average solution value increases.
  - Spread of solution values decreases.



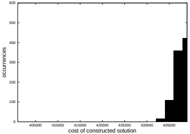
$\alpha = 0$ (random)



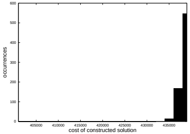
$\alpha = 0.2$



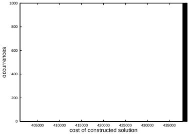
$\alpha = 0.4$



$\alpha = 0.6$



$\alpha = 0.8$



$\alpha = 1$ (greedy)

GRASP – Greedy Randomized Adaptive Search Procedures

- A greedy randomized adaptive search procedure (GRASP) is the hybridization of a semi-greedy algorithm with a local search method – embedded in a multistart framework.

```
begin GRASP;
  f* ← ∞;
  while stopping criterion not satisfied do
      S ← SEMI-GREEDY;
      if S is not feasible then
          S ← REPAIR(S);
      end-if;
      S ← LOCAL-SEARCH(S);
      if f(S) < f* then
          S* ← S;
          f* ← f(S);
      end-if;
  end-while;
  return S*;
end GRASP.
```

GRASP – Greedy Randomized Adaptive Search Procedures

- A greedy randomized adaptive search procedure (GRASP) is the hybridization of a semi-greedy algorithm with a local search method – embedded in a multistart framework.
- The method consists of multiple applications of local search, each starting from a solution generated with a semi-greedy construction procedure.

```
begin GRASP;
  f* ← ∞;
  while stopping criterion not satisfied do
      S ← SEMI-GREEDY;
      if S is not feasible then
          S ← REPAIR(S);
      end-if;
      S ← LOCAL-SEARCH(S);
      if f(S) < f* then
          S* ← S;
          f* ← f(S);
      end-if;
  end-while;
  return S*;
end GRASP.
```

GRASP – Greedy Randomized Adaptive Search Procedures

- A greedy randomized adaptive search procedure (GRASP) is the hybridization of a semi-greedy algorithm with a local search method – embedded in a multistart framework.
- The method consists of multiple applications of local search, each starting from a solution generated with a semi-greedy construction procedure.
- If the constructed solution is infeasible, a repair procedure may be needed to make it feasible.

```
begin GRASP;
  f* ← ∞;
  while stopping criterion not satisfied do
      S ← SEMI-GREEDY;
      if S is not feasible then
          S ← REPAIR(S);
      end-if;
      S ← LOCAL-SEARCH(S);
      if f(S) < f* then
          S* ← S;
          f* ← f(S);
      end-if;
  end-while;
  return S*;
end GRASP.
```

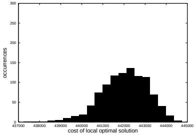
GRASP – Greedy Randomized Adaptive Search Procedures

- A greedy randomized adaptive search procedure (GRASP) is the hybridization of a semi-greedy algorithm with a local search method – embedded in a multistart framework.
- The method consists of multiple applications of local search, each starting from a solution generated with a semi-greedy construction procedure.
- If the constructed solution is infeasible, a repair procedure may be needed to make it feasible.
- A best local optimum, over all GRASP iterations, is returned as the solution provided by the algorithm.

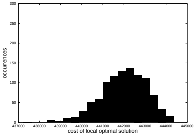
begin GRASP;
  $f^* \leftarrow \infty$;
  while stopping criterion not satisfied **do**
    $S \leftarrow$ SEMI-GREEDY;
    if S is not feasible **then**
      $S \leftarrow$ REPAIR(S);
    end-if;
    $S \leftarrow$ LOCAL-SEARCH(S);
    if $f(S) < f^*$ **then**
      $S^* \leftarrow S$;
      $f^* \leftarrow f(S)$;
    end-if;
  end-while;
  return S^*;
end GRASP.

Distribution of the solution values obtained after local search as a function of the quality-based parameter α of the semi-greedy construction procedure (1000 repetitions for each value of α) on an instance of max-SAT.

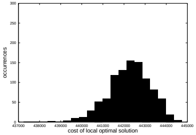
- The distributions show:



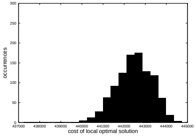
$\alpha = 0$ (random)



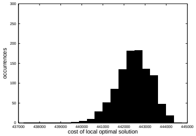
$\alpha = 0.2$



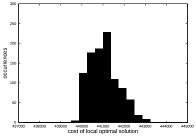
$\alpha = 0.4$



$\alpha = 0.6$



$\alpha = 0.8$

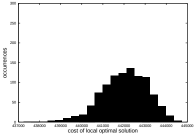


$\alpha = 1$ (greedy)

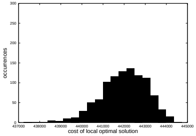
GRASP

Distribution of the solution values obtained after local search as a function of the quality-based parameter α of the semi-greedy construction procedure (1000 repetitions for each value of α) on an instance of max-SAT.

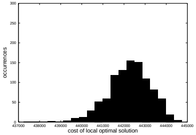
- The distributions show:
  - The variance of the GRASP solution values decreases as α increases.
  - GRASP solutions improve on average as we move from a totally random construction to a greedy construction.
  - GRASP and semi-greedy multistart differ in one important way. The best solution found, over all 1000 runs, improves as we move from random to semi-greedy construction (until some value of parameter α), and then deteriorates as α approaches 1.



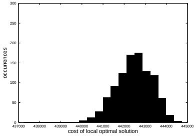
$\alpha = 0$ (random)



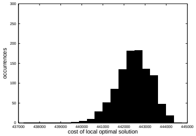
$\alpha = 0.2$



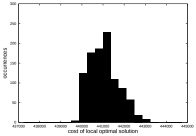
$\alpha = 0.4$



$\alpha = 0.6$



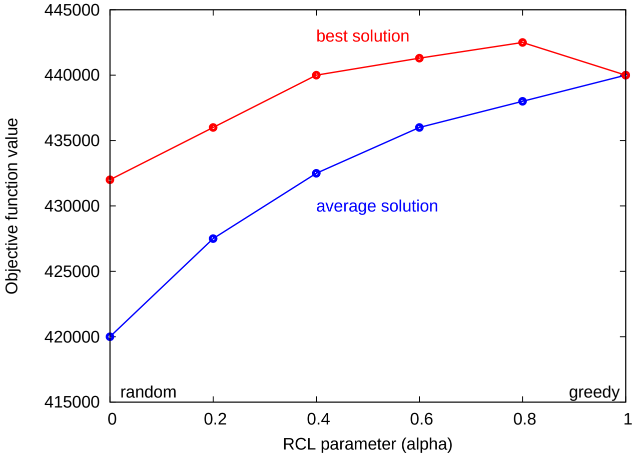
$\alpha = 0.8$



$\alpha = 1$ (greedy)

GRASP

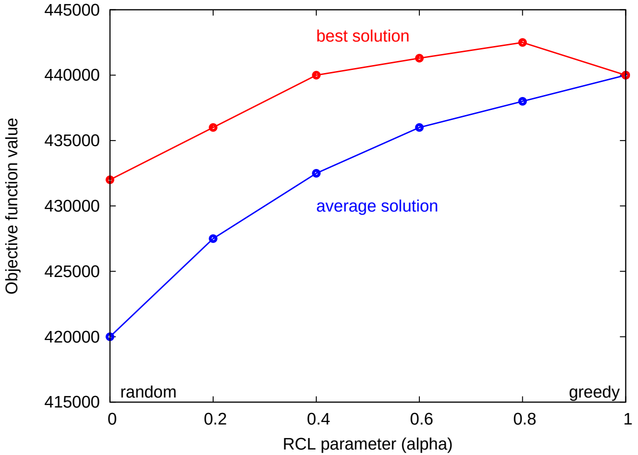
The plot shows best and average solution values for GRASP as a function of the RCL parameter α for 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.



GRASP

The plot shows best and average solution values for GRASP as a function of the RCL parameter α for 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.

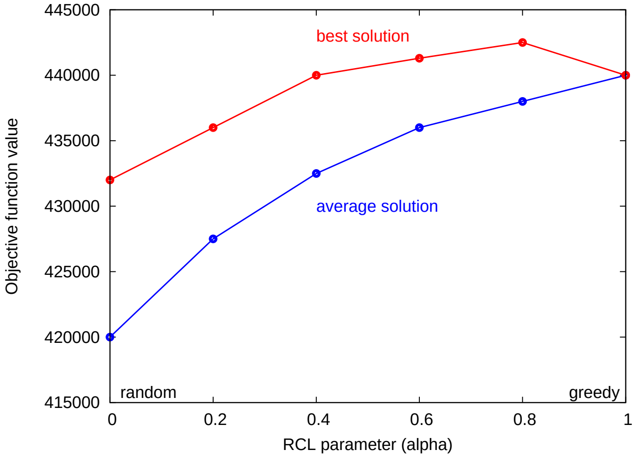
- Average semi-greedy solution improves as we move from a more random construction to a more greedy construction.



GRASP

The plot shows best and average solution values for GRASP as a function of the RCL parameter α for 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.

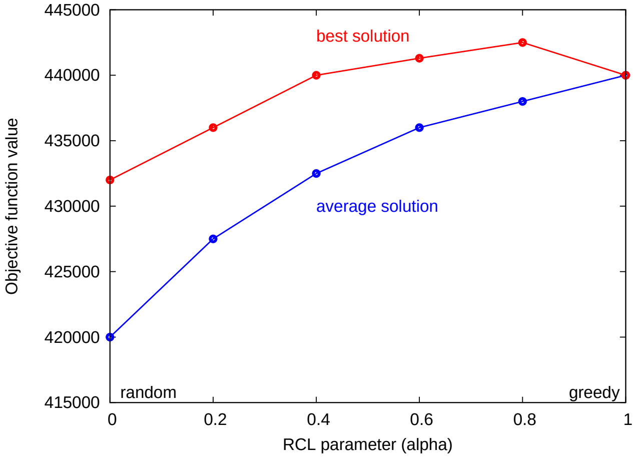
- Average semi-greedy solution improves as we move from a more random construction to a more greedy construction.
- Best solution increases, reaches a maximum, and then deteriorates.



GRASP

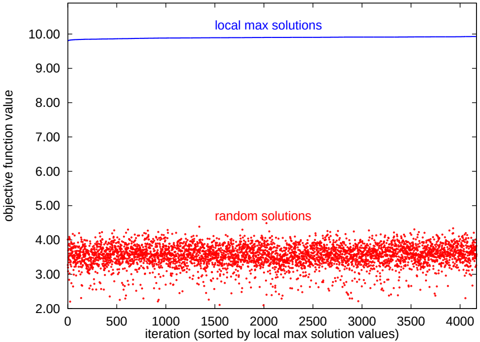
The plot shows best and average solution values for GRASP as a function of the RCL parameter α for 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.

- Average semi-greedy solution improves as we move from a more random construction to a more greedy construction.
- Best solution increases, reaches a maximum, and then deteriorates.
- Greediness is nice, but we need some diversity.

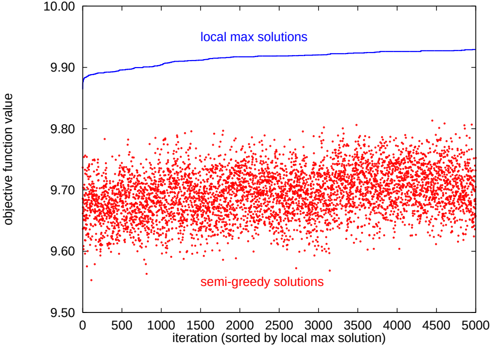


GRASP

Constructed and local maximum solution values, sorted by local maximum values, for an instance of the maximum covering problem.



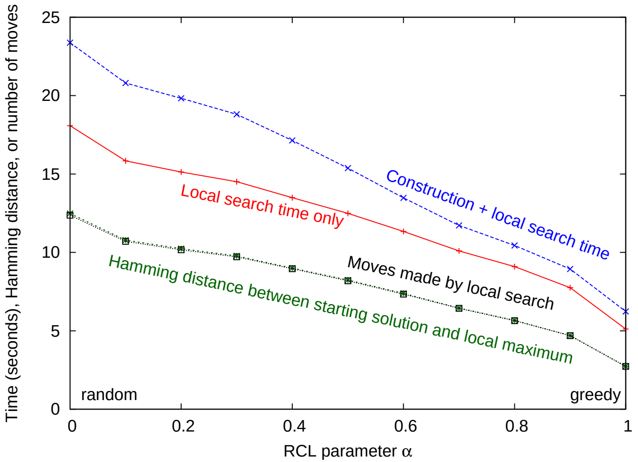
Random construction



Semi-greedy construction (RCL parameter $\alpha = 0.85$)

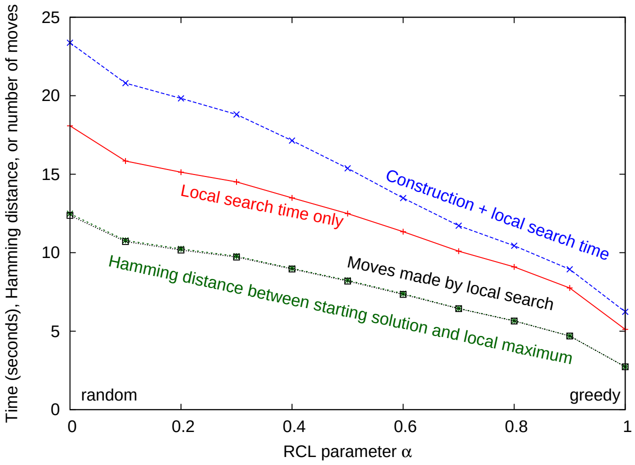
GRASP

Plot shows **total GRASP running time**, **total local search running time**, **average Hamming distance between constructed solution and local maximum**, and **average number of local search moves** as a function of the RCL parameter α on 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.



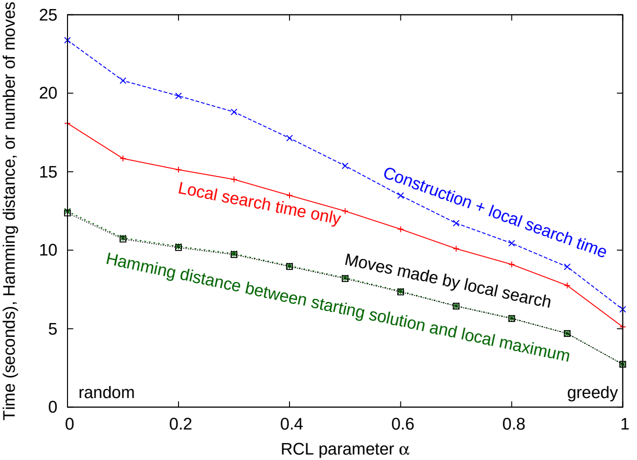
Plot shows **total GRASP running time**, **total local search running time**, **average Hamming distance between constructed solution and local maximum**, and **average number of local search moves** as a function of the RCL parameter α on 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.

- Since local search traverses a 1-flip neighborhood, the curve for the number of moves made by local search coincides with the curve for the Hamming distance between the starting solution and the local maximum.



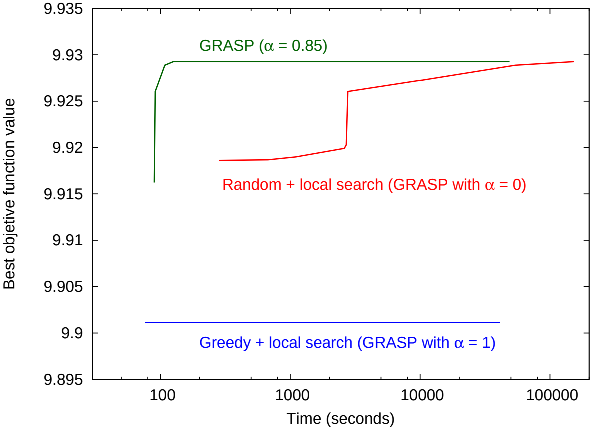
Plot shows **total GRASP running time**, **total local search running time**, **average Hamming distance between constructed solution and local maximum**, and **average number of local search moves** as a function of the RCL parameter α on 1000 GRASP iterations on an instance of the maximum weighted satisfiability problem.

- Since local search traverses a 1-flip neighborhood, the curve for the number of moves made by local search coincides with the curve for the Hamming distance between the starting solution and the local maximum.
- Strong correlation between Hamming distance, number of moves taken by local search, and local search running time.



GRASP vs random multistart vs greedy multistart

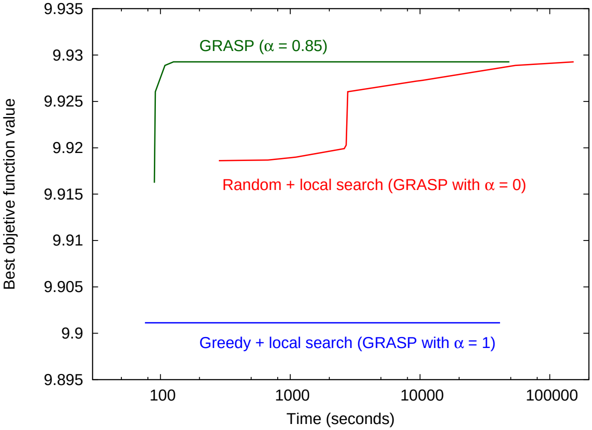
The figure displays, for the same instance of the maximum covering problem considered earlier, the best objective function solution value as a function of running time for GRASP (with $\alpha = 0.85$), random multistart (GRASP with $\alpha = 0$) with local search, and greedy multistart (GRASP with $\alpha = 1$) with local search.



GRASP vs random multistart vs greedy multistart

The figure displays, for the same instance of the maximum covering problem considered earlier, the best objective function solution value as a function of running time for GRASP (with $\alpha = 0.85$), random multistart (GRASP with $\alpha = 0$) with local search, and greedy multistart (GRASP with $\alpha = 1$) with local search.

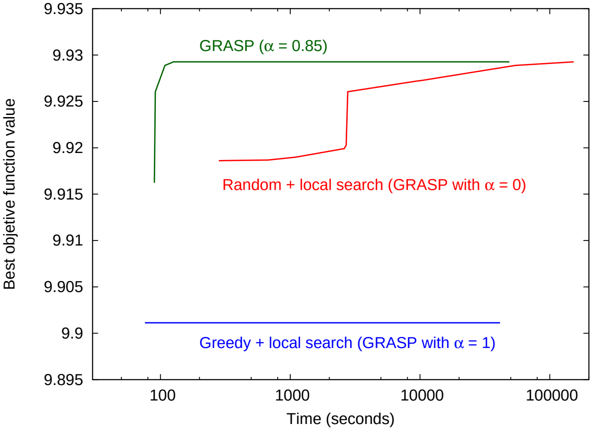
- Greedy multistart with local search fails to find the best known solution of value 9.92926.



GRASP vs random multistart vs greedy multistart

The figure displays, for the same instance of the maximum covering problem considered earlier, the best objective function solution value as a function of running time for GRASP (with $\alpha = 0.85$), random multistart (GRASP with $\alpha = 0$) with local search, and greedy multistart (GRASP with $\alpha = 1$) with local search.

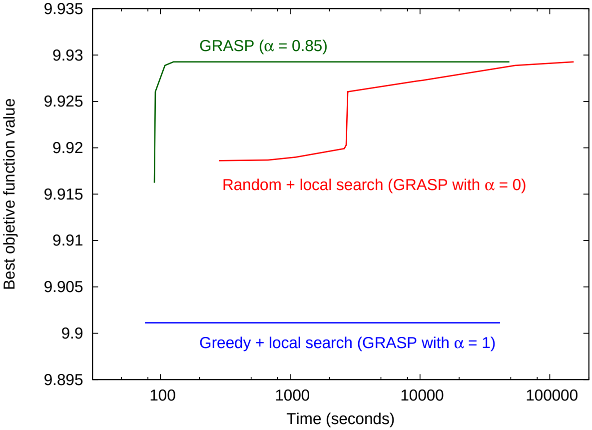
- Greedy multistart with local search fails to find the best known solution of value 9.92926.
- GRASP finds it after only 126 seconds.



GRASP vs random multistart vs greedy multistart

The figure displays, for the same instance of the maximum covering problem considered earlier, the best objective function solution value as a function of running time for GRASP (with $\alpha = 0.85$), random multistart (GRASP with $\alpha = 0$) with local search, and greedy multistart (GRASP with $\alpha = 1$) with local search.

- Greedy multistart with local search fails to find the best known solution of value 9.92926.
- GRASP finds it after only 126 seconds.
- Random multistart with local search takes 152,664 seconds to reach that solution, i.e. over one thousand times longer than GRASP.



Example of GRASP for maximum weighted cut of a graph

Given

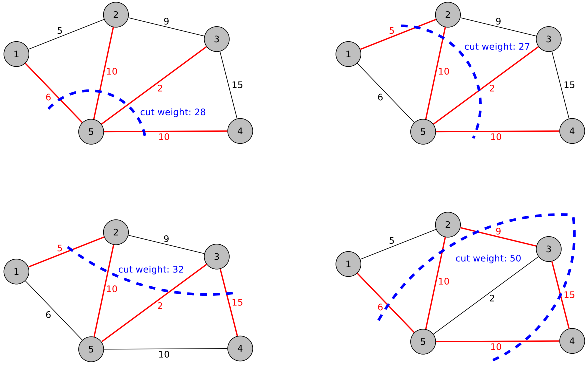
- Graph $G = (V, U)$, where V is the set of vertices and U is the set of edges;
- Weights w_{uv} associated with each edge $(u, v) \in U$.

The *maximum cut* (MAX-CUT) *problem* consists in finding a nonempty proper subset of vertices $S \subset V$ ($S \neq \emptyset$), such that the weight of the cut $(S, \bar{S})$, given by

$$w(S, \bar{S}) = \sum_{u \in S, v \in \bar{S}} w_{uv},$$

is maximized.

MAX-CUT is NP-hard (Karp, 1972).



Maximum cut problem on a graph with $|V| = 5$ and $|U| = 7$. Four cuts are shown. The maximum cut is $(S, \bar{S}) = (\{1, 2, 4\}, \{3, 5\})$ and has a weight $w(S, \bar{S}) = 50$.

GRASP for MAX-CUT

Pseudo-code of a GRASP for the MAX-CUT problem is shown on the right.

GRASP iterations continue until a stopping criterion is satisfied. Each iteration of GRASP consists in:

- Construction of a semi-greedy solution $(S, \bar{S})$ in line 3.
- Local search for a local maximum $(S, \bar{S})$ in line 4.
- Update of the best solution $(S^*, \bar{S}^*)$ in lines 7 and 8 if current local maximum is best so far.

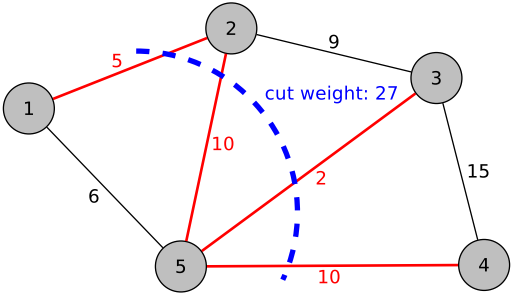
begin GRASP-MAXCUT;
1 $w^* \leftarrow -\infty$;
2 **while** stopping criterion is not satisfied **do**
3 $\quad (S, \bar{S}) \leftarrow$ SEMI-GREEDY-MAXCUT;
4 $\quad (S, \bar{S}) \leftarrow$ LOCAL-SEARCH-MAXCUT$((S, \bar{S}))$;
5 $\quad w(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} w_{ij}$;
6 $\quad$ **if** $w(S, \bar{S}) > w^*$ **then**
7 $\quad\quad (S^*, \bar{S}^*) \leftarrow (S, \bar{S})$;
8 $\quad\quad w^* \leftarrow w(S, \bar{S})$;
9 $\quad$ **end-if**;
10 **end-while**;
11 **return** $(S^*, \bar{S}^*), w^*$;
end GRASP-MAXCUT.

Greedy algorithm for MAX-CUT

We wish to build a proper subset $S \subset V$, such that $(S, \bar{S})$ forms a partition of V, i.e., $S \cup \bar{S} = V$ and $S \cap \bar{S} = \varnothing$.

The ground set for the MAX-CUT problem is the set V of vertices of graph $G = (V, U)$.

- The greedy algorithm builds a solution incrementally in sets X and Y by assigning vertices from the ground set V to either X or Y.
- Initially, sets X and Y each contain an endpoint of a largest-weight edge.
- At each other step of the construction, a new ground set element $v \in V$ is added to either set X or set Y of the partial solution.
- This is repeated until $X \cup Y = V$, at which point we set S to X, $\bar{S}$ to Y, and a feasible solution $(S, \bar{S})$ is on hand.



Cut $(S, \bar{S}) = (\{1, 5\}, \{2, 3, 4\})$ with weight 27.

Greedy algorithm for MAX-SAT

- While $|X| + |Y| < |V|$:
  - Let (X, Y) be the partial solution under construction. For each yet-unassigned vertex $v \in V \setminus (X \cup Y)$, define
  $$\sigma_X(v) = \sum_{u \in Y} w_{vu}$$
  and
  $$\sigma_Y(v) = \sum_{u \in X} w_{vu}$$
  to be, respectively, the incremental contributions to the cut weight resulting from the assignment of node v to sets X and Y of the partial partition (X, Y).
  - The greedy function
  $$g(v) = \max\{\sigma_X(v), \sigma_Y(v)\},$$
  for $v \in V \setminus (X \cup Y)$, measures how much additional weight results from the assignment of vertex v to X or Y. The greedy choice is
  $$v^* = \text{argmax}\{g(v) \ : \ v \in V \setminus (X \cup Y)\}.$$
  Vertex v^* is assigned to set X if $\sigma_X(v) > \sigma_Y(v)$ or to set Y, otherwise.

MAX-CUT: Local search

- Since a solution $(S, \bar{S})$ generated with a semi-greedy algorithm is not guaranteed to be locally optimum with respect to any neighborhood structure, a local search algorithm may improve its weight.
- To each vertex $v \in V$, we associate either
  - Neighbor $(S \setminus \{v\}, \bar{S} \cup \{v\})$ if $v \in S$
  - Neighbor $(S \cup \{v\}, \bar{S} \setminus \{v\})$ if $v \in \bar{S}$

  In other words, we move vertex v from one side of the cut to the other.

begin LOCAL-SEARCH-MAXCUT($(S, \bar{S})$);
  change $\leftarrow$ .TRUE.;
  while *change* **do**
    change $\leftarrow$ .FALSE.;
    for $v = 1, \ldots, |V|$ **while** .NOT.*change* **do**
      if $v \in S$ **and** $\sigma_{\bar{S}}(v) - \sigma_S(v) > 0$ **then**
        $S \leftarrow S \setminus \{v\}$;
        $\bar{S} \leftarrow \bar{S} \cup \{v\}$;
        change $\leftarrow$ .TRUE.;
      else
        if $v \in \bar{S}$ **and** $\sigma_S(v) - \sigma_{\bar{S}}(v) > 0$ **then**
          $\bar{S} \leftarrow \bar{S} \setminus \{v\}$;
          $S \leftarrow S \cup \{v\}$;
          change $\leftarrow$ .TRUE.;
        end-if;
      end-if;
    end-for;
  end-while;
  return $(S, \bar{S})$, $w(S, \bar{S})$;
end LOCAL-SEARCH-MAXCUT.

MAX-CUT: Local search

Let

$$\sigma_S(v) = \sum_{u \in \bar{S}} w_{vu}$$

be the sum of the weights of the edges incident to v that have their other endpoint in $\bar{S}$ and

$$\sigma_{\bar{S}}(v) = \sum_{u \in S} w_{vu}.$$

be the sum of the weights of the edges incident to v that have their other endpoint in S. The value

$$\delta(v) = \begin{cases} \sigma_{\bar{S}}(v) - \sigma_S(v), & \text{if } v \in S, \\ \sigma_S(v) - \sigma_{\bar{S}}(v), & \text{if } v \in \bar{S}, \end{cases}$$

represents the change in the objective function

If change is positive, i.e. if $\delta(v) > 0$, then make move.

```
begin LOCAL-SEARCH-MAXCUT((S, S̄));
  change ← .TRUE.;
  while change do
     change ← .FALSE.;
     for v = 1, ..., |V| while .NOT.change do
        if v ∈ S and σ_S̄(v) − σ_S(v) > 0 then
           S ← S \ {v};
           S̄ ← S̄ ∪ {v};
           change ← .TRUE.;
        else
           if v ∈ S̄ and σ_S(v) − σ_S̄(v) > 0 then
              S̄ ← S̄ \ {v};
              S ← S ∪ {v};
              change ← .TRUE.;
           end-if;
        end-if;
     end-for;
  end-while;
  return (S, S̄), w(S, S̄);
end LOCAL-SEARCH-MAXCUT.
```

Concluding remarks – Advancing GRASP

Over the years, many people have contributed to the advancement of GRASP. Here is a sample:

Concluding remarks – Advancing GRASP

Over the years, many people have contributed to the advancement of GRASP. Here is a sample:

Algorithmic developments

- GRASP with path-relinking –
  memory, intensification, improve search
- extended construction mechanisms
- Runtime distribution of GRASP –
  parallel GRASP, restart strategies
- Solution value distribution of GRASP –
  stopping GRASP
- Multi-objective GRASP
- Continuous GRASP

Concluding remarks – Advancing GRASP

Over the years, many people have contributed to the advancement of GRASP. Here is a sample:

Algorithmic developments

- GRASP with path-relinking – memory, intensification, improve search
- extended construction mechanisms
- Runtime distribution of GRASP – parallel GRASP, restart strategies
- Solution value distribution of GRASP – stopping GRASP
- Multi-objective GRASP
- Continuous GRASP

Applied GRASP

- Graph planarization
- Jet ink printer nozzle design at HP
- Locating modem pools at AT&T
- Identifying communities of interest in massive telephone call graphs
- Handover minimization in celular networks at AT&T
- Jet engine blade balancing
- In-bound baggage handling at airports
- Private virtual circuit routing
- Scheduling football tournaments

Concluding remarks

The material in this talk is taken from

- Chapter 3 – Solution construction and greedy algorithms
- Chapter 5 – GRASP: The basic heuristic
- Chapter 12 – Case studies

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