GRASP

Greedy Randomized Adaptive
Search Procedure

Type of problems

- Combinatorial optimization problem:

Finite ensemble $E = \{1,2,....n\}$

Subset of feasible solutions $F \subseteq 2^E$

Objective function $f : 2^E \rightarrow \Re$

Minimisation Problem : $S^* \in F, \ f(S^*) \leq f(S)$

Multi start meta heuristic

- -Create multiple initial solutions (construction)
- -Repair solution if it is infeasible (repair)
- -Run a local search in its neighborhood (local search)
- -Iterate over these three steps until stopping condition is met

History

- Concept was formalized by Feo and Resende.
- Article published in 1989

GRASP overview

- Most of the literature is base on the construction phase and improvements to the construction phase

- GRASP can be used with any combination of construction algorithms and local search algorithms

Plan

- Construction phase
- Improvements to the construction phase
- GRASP extensions:
  - Path relinking
  - Overview of a few others

- Parallel GRASP

Construction Phase

- Initial solutions are built iteratively by adding elements.

- At each stage of the initial solution construction, a Restricted Candidate List (RCL) is constructed.

Construction continued

- The RCL contains an ordering of the best next element to add to the solution.
- The next element is chosen from a truncation of this list
- Iterate until we have an initial solution

Construction algorithm

```
procedure Greedy Randomized Construction(Seed)
      Solution← /0;
      Initialize the set of candidate elements;
      Evaluate the incremental costs of the candidate elements;
      while there exists at least one candidate element do
            Build the restricted candidate list (RCL);
            Select an element s from the RCL at random;
            Solution←Solution+{s};
            Update the set of candidate elements;
            Reevaluate the incremental costs;
end;
return Solution;
end Greedy Randomized Construction.
```

RCL List construction

- Rank approach
  - Keep the p best elements in the RCL.
  - Take random element from RCL
  - If $p \rightarrow N$, purely random construction
  - If $p \rightarrow 1$, purely greed construction (and only one determinist initial solution)

RCL Continued

- Relative Threshold Approach
  - Only keep elements in the RCL that are worst than best element by a factor

element $e \in E$

cost $c(e)$

Only keep elements that obey $: c(e) \in [c^{\min}, c^{\min} + \alpha(c^{\max} - c^{\min})]$

  - Alpha is a hyper parameter

Alpha→0, purely greedy algorithm

Alpha→1, purely random algorithm

RCL tradeoff

- In both cases, balance must occur between purely random (multiple average initial solutions) and purely greedy (only one solution is tried).
- Also, computation cost of local search is inversely proportional to quality of initial solution and computation cost of local search > initial solution construction cost

RCL Tradeoff

Table 1 Average number of moves and local search time as a function of the RCL parameter α for a maximization problem.

α	avg. distance	avg. moves	local search time (s)	total time (s)
0.0	12.487	12.373	18.083	23.378
0.1	10.787	10.709	15.842	20.801
0.2	10.242	10.166	15.127	19.830
0.3	9.777	9.721	14.511	18.806
0.4	9.003	8.957	13.489	17.139
0.5	8.241	8.189	12.494	15.375
0.6	7.389	7.341	11.338	13.482
0.7	6.452	6.436	10.098	11.720
0.8	5.667	5.643	9.094	10.441
0.9	4.697	4.691	7.753	8.941
1.0	2.733	2.733	5.118	6.235

RCL Tradeoff

- Typical alpha~0.2

Classic construction procedure shortcomings

- RCL construction can be expensive since it is constructed for every element added to the initial solution

- No history of initial solution is kept
  - Past good initial solutions should influence construction of future solution

Construction phase improvements

- Many algorithms exist to improve construction phase (both for quality and complexity)
  - Complexity reduction
  - Reactive Grasp
  - Bias functions
  - Intelligent construction
  - POP principle
  - Cost perturbation

Complexity Reduction

- Random + greedy
  - Randomly add first p elements to the initial solution
  - Complete the solution with greedy approach
- Sampled greedy
  - Only sample a few elements to put in RCL instead of all remaining elements

Reactive Grasp

- Variable Alpha
- At each new initial solution construction phase, choose Alpha from a set of possible Alpha.
- The probability of choosing a given Alpha is proportional to the quality of the past solutions with this Alpha

Reactive GRASP continued

- Formally:

Set of possible Alpha :	$\psi = \{\alpha\ , \alpha_2, \ldots \alpha_m\}$
Probability of each Alpha :	p_i
Cost of incumbent solution :	z^*
Average value of solutions with α_i :	A_i
Relative score of a given α_i :	$q_i = \frac{z^*}{A_i}$
Updated Probabilty for α_i :	$\frac{q_i}{\sum q_i}$

Bias functions

- Next element to take from the RCL to add to solution under construction is usually chosen randomly from RCL
- Use a bias function based on rank so choice is no longer uniform random (same as gene selection through rank)

Bias function continued

- A few proposed bias functions proposed:

$$\text{uniform } bias(r) = 1$$

$$\text{linear } bias(r) = 1/r$$

$$\text{exponential } bias(r) = e^{-r}$$

$$\text{polynomial} bias(r) = r^{-n}$$

$$\text{probability of element } \sigma \text{ of RCL } = \frac{bias(r(\sigma))}{\sum bias(r(\sigma))}$$

Intelligent construction

- Keep a pool of elite solution
- This pool contains the <u>best</u> solutions that are sufficiently <u>different</u> between each other
- When constructing an initial solution, choose elements from RCL that will give a solution which contains patterns in elite solutions

Intelligent construction

- Example in TSP:
- Assume elite solutions contain the same 3 linked nodes
- The construction phase should favor the use of these same three nodes so they are linked

Proximate Optimality Principle (POP)

- Greedy construction is not always optimal. Especially on huge problems.
- Instead, run a few local search algorithms during construction phase
- Ex: Binato ran a local search at 40% and 80% during construction

Cost Perturbation

- Modify cost function when building RCL based on past solutions.
- Not necessarily applicable to every problem.
- Good results obtained using this for the Steiner tree problem (interconnect points on a plane)

Path Re-linking

- Given two solutions, explore the path linking these two solutions in the solution space.
- Start from one solution, go to best neighbor that brings us closer to the other solution. Iterate until reaching other solution.

Path re-linking

- Formally:

calculate moves necessary to transform s_1 into s_2 $:\Delta(s_1, s_2)$

while $\mathrm{s}_1 != s_2$

evaluate cost of $\mathrm{s}_i' = \mathrm{s}_1 + \delta_i, \quad \delta_i \in \Delta(s_1, s_2), \qquad \forall i$

$s_1 = \min(s_i)$

if($\mathrm{s}_1 <$ global best)

global best $= \mathrm{s}_1$

end

Path re-linking and GRASP

- Multiple ways of incorporating path re-linking with GRASP.
- Most keep a pool of elite solutions. As before, this pool contains the <u>best</u> solutions that are sufficiently <u>different</u> between each other

Path re-linking and GRASP

- Strategies:
  - Intensification: re-link all local solution with one or more elite solution
  - Intensification: re-link elites between each other periodically (akin to an evolutionary process)
  - Post-optimization: re-link every elite solution
  - Post-optimization: submit pool to an evolutionary process

Intensification

- Find a local minima s through local search
- Choose a solution g from elite pool. g should be chosen so that it is different than s (different hamming distance for example).
- Determine which solution is the initial solution and which is the destination

Intensification continued

- Apply one of the path re-linking algorithms

Intensification variations

- Forward path re-linking
  - Go from local minima to elite solution
- Backward path re-linking
  - Go from elite solution to local minima
    - (more logical since elite neighborhood should be on average better)
- Back and forward path re-linking
    - Do both. Expensive but best of both solutions.

Intensification variations

- Mixed path re-linking
  - Ping pong between local minima and elite solution.
  - Should give the advantages of back and forward at half the cost
- Truncated path re-linking
  - Only explore close neighborhood of elite solution and/or local minima

Intensification variations

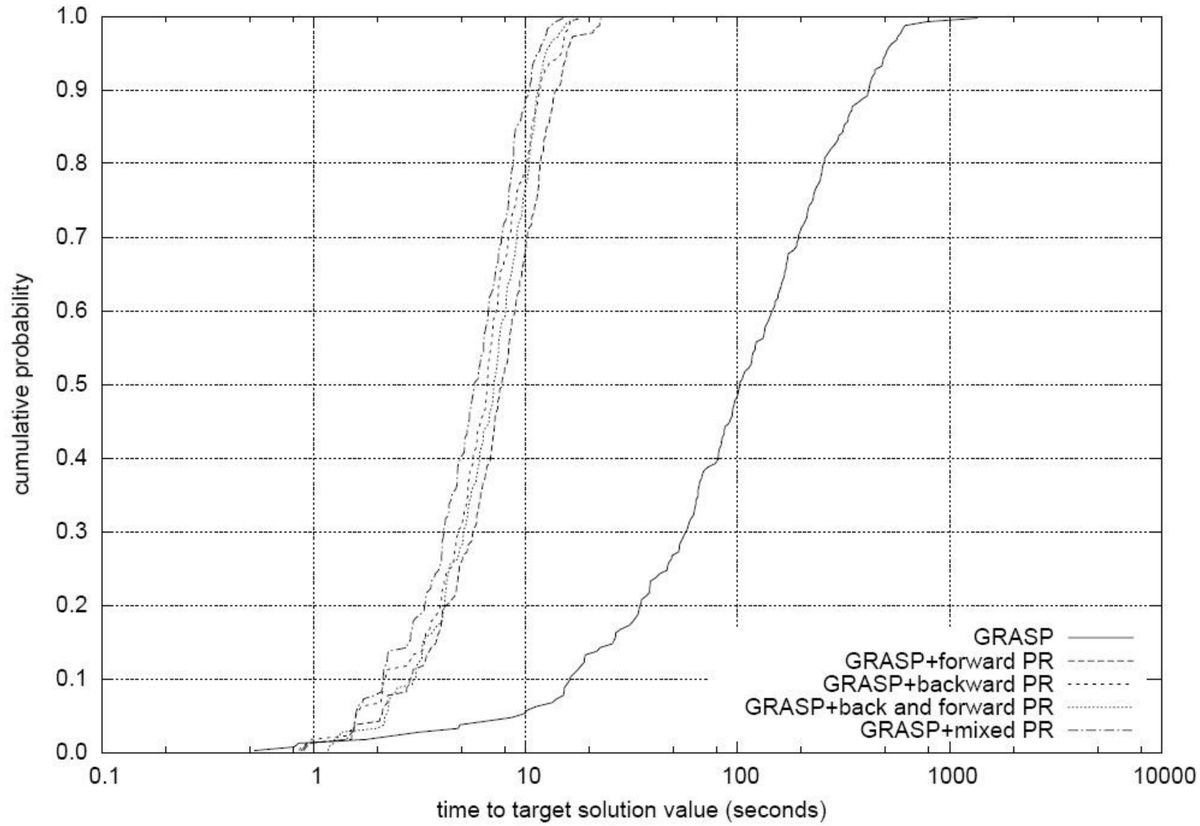
- Greedy randomized adaptive path re-linking
  - Number of paths between solution is exponential in $|\Delta(s_1, s_2)|$
  - Instead of greedily exploring one path, occasionally choose the next element randomly
    - Leads to more path being explored and diversifies the paths explored

Intensification variations

- Evolutionary path re-linking
  - Every few GRASP iteration, re-link the solutions in the elite pool to create a new population of elites.
  - Use greedy randomized path re-linking to diversify the elite pool

Path re-linking advantages

- Path re-linking helps reduce the time before a target optimum is reached



GRASP extensions and implementation ideas

- Use of a hash table of already explored solution can limit wasting time exploring multiple times the same solution neighborhood.
- Not all initial solutions need to be explored. Only promising ones should be investigated

GRASP hybrids

- Classic GRASP uses a simple local search. But it can use a VNS instead as the two can be seen as complimentary (construction vs. exploration). Tabu search can also be used. In fact, any search meta heuristic.

- Grasp construction phase can also be used to create an initial population for genetic algorithms.

Grasp algorithm comparison

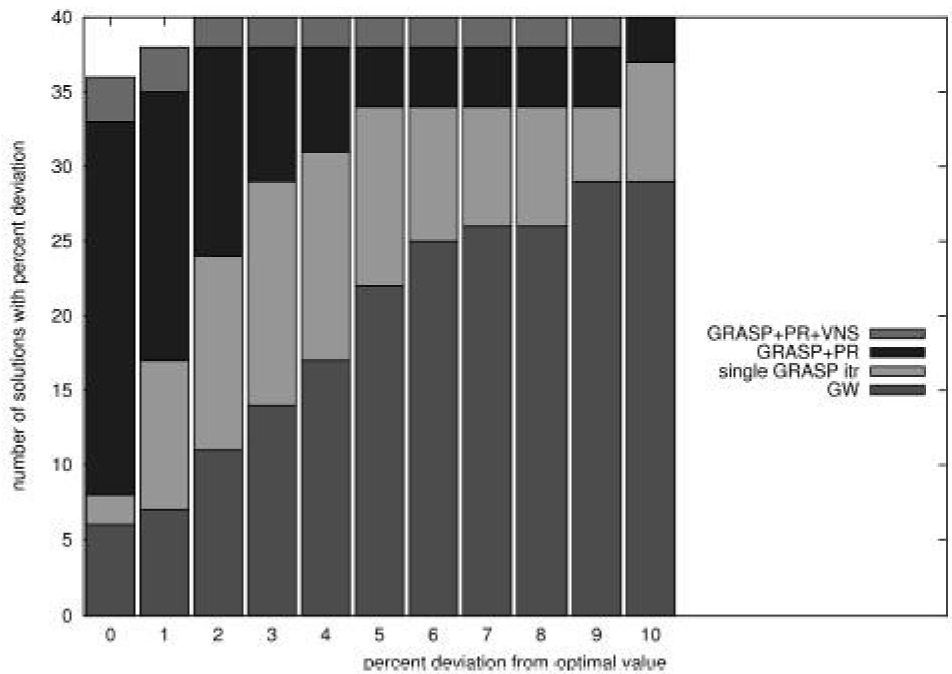


Fig. 12 Performance of GW approximation algorithm, a single GRASP iteration (GW followed by local search), 500 iterations of GRASP with path-relinking, and 500 iterations of GRASP with path-relinking followed by VNS for series C prize-collecting Steiner tree problems.

Parallel Grasp

- Multi start heuristic particularly well suited for parallel implementation, running on multiple CPUs.
- Two types of strategies:
  - Multiple walk, independent thread
  - Multiple walk, cooperative thread

Independent thread

- Each CPU runs a subset of iterations of GRASP. Best solution of all CPUs is kept. CPUs do not exchange any other info.
- Linear gain in time.
- Does not work with intensification strategies and path re-linking

Cooperative thread

- Same principle but CPUs share info.
- Info can be pool of elites, Alpha bias functions etc.
- Usually, all shared info is managed by a central processor.

Cooperative vs. independent

- Cooperative is better with many CPUs as we loose 1 CPU for info sharing.

Conclusion

- GRASP is a pretty simple concept to implement
- Easily modified by changing construction algorithm or local search algorithm
- Low numbers of hyper parameter to tune
- Efficient parallel implementation