GRASP – A speedy introduction

Thomas Stidsen
thst@man..dtu.dk

DTU-Management
Technical University of Denmark
GRASP

- GRASP is an abbreviation for
  - Greedy
  - Randomized
  - Adaptive
  - Search
  - Procedure.
- It was “invented” by Feo and Resende in 1989.
Overview of GRASP

GRASP consists of 2 phases:

- Greedy Randomized Adaptive phase (generating a solution).
- Local Search (finding a local minimum).
Greedy Randomized Adaptive phase

The first phase consists of 3 parts:

1. Greedy algorithm.
Greedy algorithm

- A greedy algorithm always makes the choice that looks best at the moment. It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

- Greedy algorithms do not always yield optimal solutions (eg. 0-1-knapsack), but in some cases it does (eg. Minimum spanning tree).
Mathematical Model for Knapsack problem

**objective:** maximize

\[
\sum_{i} c_i \cdot x_i
\]

**s.t.**

\[
\sum_{i} w_i \cdot x_i \leq W
\]

\[x_i \in \{0, 1\}\]

The profit weight ratio: \[PF = \frac{c_i}{w_i}\]
Greedy algorithm for 0-1 knapsack

**procedure** greedy_knapsack()

calculate profit vs. weight ratio

**while** the smallest item can still fit in **do**

Add largest $PF = \frac{c_i}{w_i}$ element

Update remaining weight

**return** solution
Example: 0-1-Knapsack Problem

- Item 1: Value 60, Weight 10
- Item 2: Value 100, Weight 20
- Item 3: Value 120, Weight 30

Total capacity of the knapsack: 120
Example: 0-1-Knapsack Problem

knapsack 1 2 3

220
120
100

160
100
60

180
120
60

220 160 180
Greedy algorithm for the CLP?

How can we make a greedy algorithm for the Christmas lunch problem? Let's remember a couple of facts:

- We want to maximize the interest count.
- A person at a table can only achieve a positive interest from other persons at a table.
- If a table is empty and a person is put there, there is no gain...
Greedy algorithm for the CLP

So, we will make the greedy algorithm the following way:

```
procedure greedy_clp()
for each table do
    Place one un-seated person randomly chosen
while still room at the table do
    Add the person with the largest interest gain
return solution
```
Probabilistic selection

- In the greedy algorithm the selection of the next element to add to the solution is (often !) deterministic.

- The probabilistic selection process selects candidates among which we choose the next element to be added to the presently partial solution. The list of candidates is formally denoted the *Restricted Candidate List* (RCL).
Restricted Candidate List

Construction:

- Select the $\beta$ best elements ($\beta = 1$: purely greedy, $\beta = n$ completely random).
- Selects the elements that are not more than $\alpha\%$ away from the best element. ($\alpha = 0$: purely greedy, $\alpha = \infty$: purely random).
- Select elements above an absolute threshold of $\gamma$.

Selection:

- Equally probability.
- Weighted probability.
Adaptive function

- Modify the function which guides the greedy algorithm based on the element selected for the solution we are constructing.
- The construction is called **dynamic** in contrast to the **static** approach which assigns a score to elements only before starting the construction (example: TSP links).
GRASP main code

```plaintext
procedure grasp()
    CurrentBest = 0
    while ⟨ more time ⟩
        Solution = ConstructSolution()
        NewSolution = LocalSearch(Solution)
        if ⟨ NewSolution better than CurrentBest ⟩ then
            CurrentBest = NewSolution
    return CurrentBest
```
Greedy Randomized solution construction

procedure ConstructSolution()
    Solution = ∅
    for ⟨ solution construction not finished ⟩
        CandidateList = MakeCandidateList()
        s = SelectElement(CandidateList)
        Solution = Solution ∪ {s}
        ChangeGreedyFunction();
    return Solution
Additions to the GRASP scheme

- Use different adaptive functions during the execution of the algorithm.
- Settings of the Restricted Candidate List dynamic.
More information

- The website

http://www.graspheuristic.org

contains an annotated bibliography of GRASP. This is a good starting point for every form of work with the GRASP metaheuristic.

- Object-oriented framework developed by Andreatta, Carvalho and Ribero.
Pros and cons

- **PRO**: Simple structure (means often easy to implement),
- **PRO**: If you can construct a greedy algorithm, extension to GRASP is often easy.
- **PRO**: Can be used for **HUGE** optimization problems.
- **CONS**: dependent on a “natural” greedy algorithm, no escape from local optimum.
- **CONS**: may easily re-discover the same solution many times
(Adaptive) Large Neighbourhood Search

A recently developed method is: Large Neighbourhood Search (LNS) or Adaptive Large Neighbourhood Search (ALNS).

- LNS first suggested in 1998 by Shaw: "Using constraint programming and local search methods to solve vehicle routing problems".

- ALNS first suggested in 2006 by Ropke and Pisinger: "An adaptive large neighborhood search heuristic"

- Both LNS and ALNS are similar to GRASP and have shown VERY interesting results.
The basic idea in LNS is to use DESTROY and REPAIR methods to create new solutions (based on the current solution):

- A destroy method removes a part of the solution ...
- A repair method recreates the whole (feasible) solution.
The basic idea in LNS is to use DESTROY and REPAIR methods to create new solutions (based on the current solution):

- Together destroy and repair methods form a neighbourhood, typically a very large neighbourhood.
- The destroy method is often stochastic, i.e. very simply deleting a certain part of the solution.
- The repair method is very often a kind of greedy constructive algorithm.
LNS III: Pseudo code

procedure LNS()
  Generate feasible solution $x$
  $x^b = x$
  repeat
    $x^t = r(d(x))$
    if accept($x^t, x$) then $x = x^t$
    if $c(x^t) < c(x^b)$ then $x^b = x^t$
  until time limit
  return $x^b$
ALNS I

Adaptive Large Neighbourhood Search is basically an extension of LNS: Instead of having one destroy and one repair method, ALNS allows many different destroy and repair methods:

- In each iteration a destroy and repair methods is chosen probabilistically.
- The chance for being chosen is adaptively chosen so that successful repair and destroy methods are rewarded for their success.
LNS II : Pseudo code

procedure ALNS()
    Generate feasible solution $x$
    $x^b = x$, $\rho^- = (1, \ldots, 1)$, $\rho^+ = (1, \ldots, 1)$
    repeat
        select destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using $\rho^-$ and $\rho^+$
        $x^t = r(d(x))$
        if accept($x^t, x$) then $x = x^t$
        if $c(x^t) < c(x^b)$ then $x^b = x^t$
        update $\rho^-$ and $\rho^+$
    until time limit
return $x^b$
How to select destroy and repair

The probability for choosing repair/destroy operator number \( j \) is:

\[
\phi^{-} = \frac{\rho^{-}}{\sum_{k=1}^{\Omega^{-}}}
\]
How to adjust $\rho^-$ and $\rho^+$ I

For every iteration exactly one destroy and one repair method. The used methods (and only them) are given a value $\Psi = MAX(\omega_1, \omega_2, \omega_3, \omega_4)$, where $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4$.

- $\omega_1$: reward if new solution is new global best
- $\omega_2$: reward if new solution is better than the current one
- $\omega_3$: reward if new solution is accepted
- $\omega_4$: "reward" if new solution is rejected
How to adjust $\rho^-$ and $\rho^+$ II

Finally, the new $\rho^-$ and $\rho^+$ for the selected destroy and repair methods, $\rho_{a}^-$ and $\rho_{a}^+$:

$$\rho_{a}^- = \lambda \rho_{a}^- + (1 - \lambda) \rho_{a}^-$$

Where $\lambda \in [0, 1]$
LNS and ALNS comments

- Both methods have shown excellent results.
- Both methods can be considered to be generalizations of GRASP (complete destroy and probabilistic repair ...)
- A somewhat similar method "Variable Neighbourhood Search" is described in the book ... but I much prefer the handed out article by Roepke and Pisinger.