Algorithmic Methods for Mathematical Models (AMMM)

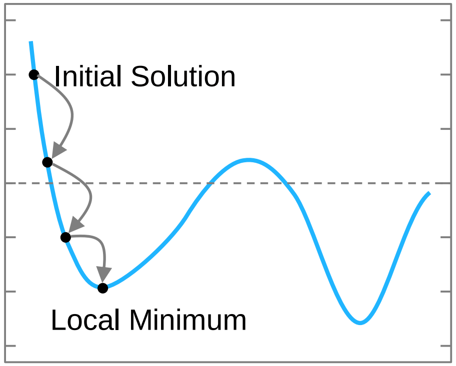
Greedy Randomized Adaptive Search Procedure (GRASP)

Luis Velasco

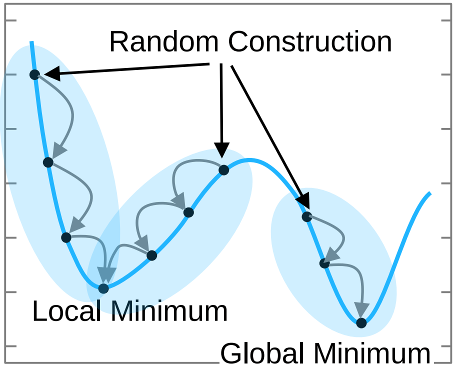
(lvelasco @ ac.upc.edu)

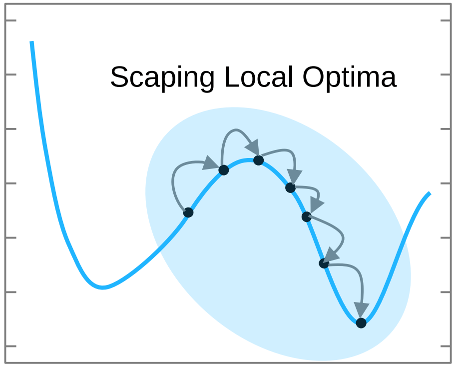
Campus Nord D6-107

Meta-heuristics



- Classical methodology limitations:
  - Deterministic
  - No global minimum is reached (just local minimum)
- Meta-heuristics go beyond heuristics by:
  - adding **variability** (randomize)
  - allowing **escaping from local optima**, at risk of cycling





Meta-heuristics

- Some well-known meta-heuristics are:
  - **GRASP** (Feo and Resende): a multi-start meta-heuristic for combinatorial problems.
  - Evolutionary algorithms (genetics). **BRKGA** (M. Resende)
  - **Simulated Annealing**, probabilistic meta-heuristic often used when the search space is discrete
  - **Tabu Search** (Fred W. Glover): Enhances the performance of local search by using memory structures.
  - **Ant colony**: probabilistic technique (Marco Dorigo)
  - **Path relinking**: an *intensification* method.

Greedy Randomized Adaptive Search Procedures (GRASP)

- GRASP* is a meta-heuristic for **combinatorial problems**.
- **Each GRASP iteration** consists basically of two phases: **construction** and **local search**.

Like in Greedy + LS

- The **construction phase** builds a feasible solution.
- Its neighborhood is investigated until a local minimum is found during the **local search** phase.
- The best overall solution is kept as the result.

$S_{best} \leftarrow \{\}$

for k=1..Max_Iterations **do**

Randomized Selection

$\boxed{S = \text{doConstructionPhase ()}}$

$\boxed{S = \text{doLocalSearch } (S)}$

Deterministic

if f(S) < f(S_{best}) **then** $S_{best} \leftarrow S$

return S_{best}

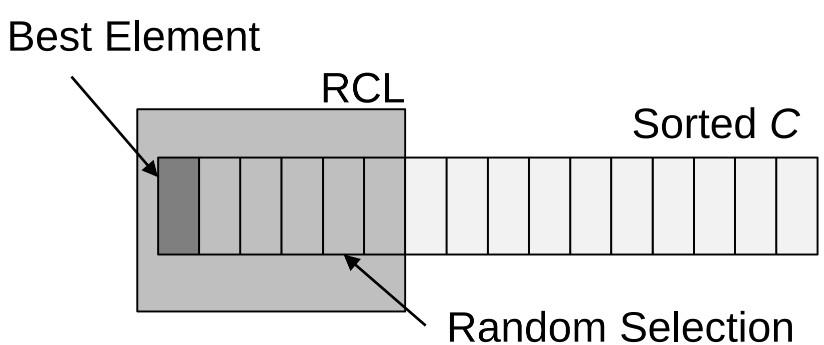
* M. Resende (http://mauricio.resende.info/)

GRASP Construction Phase

- At each iteration:
  - let the set of **candidate elements** be formed by all elements that can be incorporated to the partial solution under construction without destroying feasibility.
  - The selection of the next element to be added is determined by the evaluation of all candidate elements according to a **greedy function**.
  - A **restricted candidate list** (RCL) is created with the best elements, i.e. those whose incorporation to the current partial solution results in the **smallest incremental costs**.
- The element to be incorporated into the partial solution is **randomly selected** from those in the RCL.
- Once the selected element is incorporated to the partial solution, the candidate list is updated, and the incremental costs are reevaluated.

The Restricted Candidate List (RCL)

- The RCL contains "feasible" elements $c \in C$ that can be inserted into the partial solution without destroying feasibility.
- Elements in the RCL are those with the best (i.e., the smallest) incremental costs q(*c*).
- This list can be limited either by:
  - the **number of elements** (*cardinality-based*), the RCL is made up of the *p* elements with the best incremental costs, where *p* is a parameter.
  - their **quality** (*value-based*), the RCL is associated with a threshold parameter $\alpha \in [0,1]$.



Value-based RCL

- The RCL contains elements $c \in C$ whose quality is superior to the threshold value, i.e.,

$$q(c) \in [q^{min}, q^{min} + \alpha(q^{max} - q^{min})]$$

- The case **α=0** corresponds to a **pure greedy** algorithm
- The case **α=1** is equivalent to a **purely random** construction.

RCL (**α=0**)

9	**10**	**11**	**11**	**13**	**18**

$$q(c) \in [9, 9 + 0*(18 - 9)]$$

RCL (**α=1**)

18	**13**	**11**	**11**	**10**	**9**

$$q(c) \in [9, 9 + 1*(18 - 9)]$$

GRASP Constructive Phase

Greedy

Initialize C
$S \leftarrow \{\}$
while S is not a solution **do**
  Evaluate q(c) for all $c \in C$
  $c_{best} \leftarrow \text{argmax}\{q(c) \mid c \text{ in } C\}$
  $S \leftarrow S \cup \{c_{best}\}$
  update C
return S

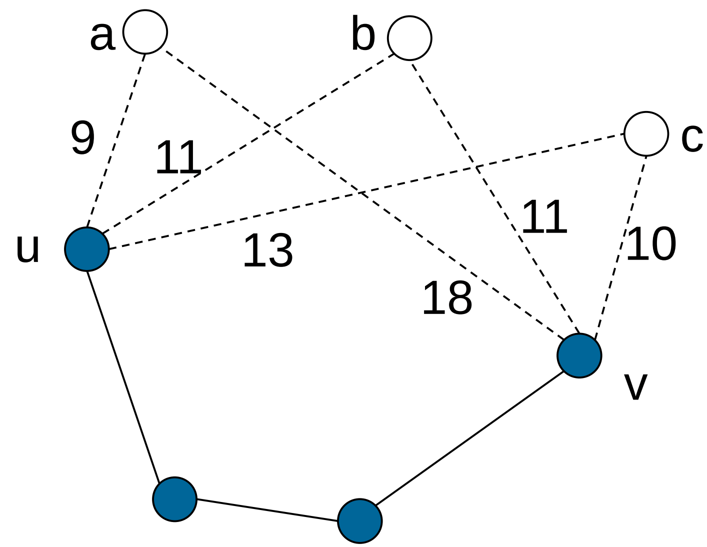
$c_{best} \leftarrow \text{argmin}\{q(c) \mid c \text{ in } C\}$

GRASP constructive phase

Initialize C
$S \leftarrow \{\}$
while S is not a solution **do**
  Evaluate q(c) for all $c \in C$
  $q^{min} \leftarrow \min \{q(c) \mid c \in C\}$
  $q^{max} \leftarrow \max \{q(c) \mid c \in C\}$
  $RCL_{max} \leftarrow \{c \in C \mid q(c) \geq q^{max} - \alpha(q^{max} - q^{min})\}$
  Select $c \in$ RCL at random
  $S \leftarrow S \cup \{s\}$
  Update C
return S

$RCL_{min} \leftarrow \{c \in C \mid q(c) \leq q^{min} + \alpha(q^{max} - q^{min})\}$

Example: TSP



α=0.3

q_{min}=9
q_{max}=18
q(e) ≤ 9+α(18-9)=11.7

RCL

9	**10**	**11**	**11**	**13**	**18**

Example: GRASP for Set Covering

M/P	p1	p2	p3	p4	p5	p6	p7	p8
1						X		
2			X	X			X	
3	X	X		X	X		X	
4	X			X	X	X		X
5					X	X		
cost	2	1	1	3	3	3	2	1

Let S the solution sub-family
Let R the set of covered elements
Greedy function:

$q(p_j)=|p_j \cap (M \backslash R)| = |p_j \backslash (R \cap p_j)| \rightarrow$ Number of additional elements of p_j

If every p_j has its own associated cost c_j, the greedy function would be:

$q(p_j) = c_j / |p_j \cap (M \backslash R)|$

We use $\boldsymbol{\alpha}$**=0.5**

Set Covering: GRASP constructive

S={} compute q(pj) ∀ $p_j \in P \setminus S$
R={} RCL={p_j | $q(c) \geq q^{max} - \alpha(q^{max} - q^{min})$}

$q^{max} = 3$, $q^{min} = 1$, $q(p_j) \geq 2$, RCL={p1, p4, p5, p6, p7}
Get one element from the RCL at random: p1
S={p1}
R={3, 4}

q(p1) = 2, q(p2) = 1, q(p3) = 1, q(p4) = 3, q(p5) = 3, q(p6) = 3, q(p7) = 2, q(p8) = 1

$q^{max} = 2$, $q^{min} = 0$, $q(p_j) \geq 1$, RCL={p3, p4, p5, p6, p7}
Get one element from the RCL at random: p6
S={p1, p6}
R={1, 3, 4, 5}

q(p2) = 0, q(p3) = 1, q(p4) = 1, q(p5) = 1, q(p6) = 2, q(p7) = 1, q(p8) = 0

$q^{max} = 1$, $q^{min} = 0$, $q(p_j) \geq 0.5$, RCL={p3, p4, p7}
Get one element from the RCL at random: p7
S={p1, p6, p7}
R=M

q(p2) = 0, q(p3) = 1, q(p4) = 1, q(p5) = 0, q(p7) = 1, q(p8) = 0

Cost: 7

Set covering: Local Search

M/P	p1	p2	p3	p4	p5	p6	p7	p8
1						X		
2			X	X			X	
3	X	X		X	X		X	
4	X			X	X	X		X
5					X	X		
cost	2	1	1	3	3	3	2	1

GASP Constructive Solution:
S={p1, p6, p7} → Cost: 7

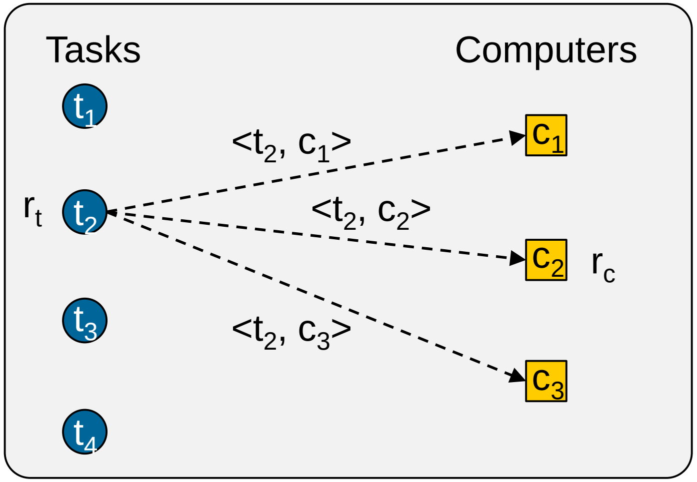
We can search **N_0(S)** and try to remove as many pj as possible from S.

M	Times covered	Covered by
1	1	{p6}
2	1	{p7}
3	2	{p1, p7}
4	2	{p1, p6}
5	1	{p6}

We can remove p1 from S, as all the elements in p1 are covered more than once

Solution after local search:
S={p6, p7} → Cost: 5

Example: Assign tasks to computers (lab session 2)



$$q(<\mathrm{t},\mathrm{c}>) = max \left\{ \begin{array}{l} 1 - \dfrac{(residualCapaciy(c) - r_t)}{r_c} \\ 1 - \dfrac{residualCapaciy(c')}{r_{c'}} \mid c' in\ C, c' \neq c \end{array} \right\}$$

GRASP constructive phase

$S \leftarrow \emptyset$
sortedT ← sort(T, r_t, *DESC*)
for each c in C **do** *residualCapacity*(c) = r_c
for each t in T **do**
    $C(t) \leftarrow \emptyset$
    for each c in C **do**
        if $r_t \leq residualCapacity(c)$ **then** $C(t) \leftarrow C(t) \cup \{c\}$
    if $|C(t)|=0$ **then return** INFEASIBLE
    $q^{min} \leftarrow \min\ \{q(<t,c>) \mid c \in C(t)\}$
    $q^{max} \leftarrow \max\ \{q(<t,c>) \mid c \in C(t)\}$
    $\mathrm{RCL}_{min} \leftarrow \{c \in C(t) \mid q(<t,c>) \leq q^{min} + \alpha(q^{max} - q^{min})\}$
    c_{sel} ← select $c \in$ RCL at random
    $residualCapacity(c_{sel}) \leftarrow residualCapacity(c_{sel}) - r_t$
    $S \leftarrow S \cup \{<t, c_{sel}>\}$
return S

Greedy

$S \leftarrow \emptyset$
sortedT ← sort(T, r_t, *DESC*)
for each c in C **do** *residualCapacity*(c) = r_c
for each t in T **do**
    $C(t) \leftarrow \emptyset$
    for each c in C **do**
        if $r_t \leq residualCapacity(c)$ **then** $C(t) \leftarrow C(t) \cup \{c\}$
    if $|C(t)|=0$ **then return** INFEASIBLE
    $c_{best} \leftarrow \mathrm{argmin}\{q(<t,c>) \mid c \text{ in } C(t)\}$
    $residualCapacity(c_{best}) \leftarrow residualCapacity(c_{best}) - r_t$
    $S \leftarrow S \cup \{<t, c_{best}>\}$
return S

Assignment Tasks to computers: Iterative execution

Computers	c1	c2	c3	
rc	505.67	503.68	701.78	
Tasks	t1	t2	t3	t4
rt	261.27	560.89	310.51	105.8

α=0.3

sortedTasks	t2	t3	t1	t4

Computers	c1	c2	c3
residualCap	505.67	503.68	701.78

#1

task: t2	560.89
C(t2)	c3
RCL	c3

Computers	c1	c2	c3
residualCap	505.67	503.68	**140.89**
load	0	0	**0.799**
S	{**<t2,c3>**}		

#2

task: t3	310.51		
C(t2)	c1	c2	
Load if assignment			
c1	0.6141	qmin	
c2	0.6165	qmax	
RCL	q≤	0.6148	{c1}

Computers	c1	c2	c3
residualCap	**195.16**	503.68	140.89
load	**0.6141**	0	0.799
S	{<t2,c3>,**<t3,c1>**}		

#3

task: t1	261.27
C(t1)	c2
Load if assignment	
c2	0.5187
RCL	c2

Computers	c1	c2	c3
residualCap	195.16	**242.41**	140.89
load	0.6141	**0.5187**	0.799
S	{<t2,c3>,<t3,c1>,**<t1,c2>**}		

#4

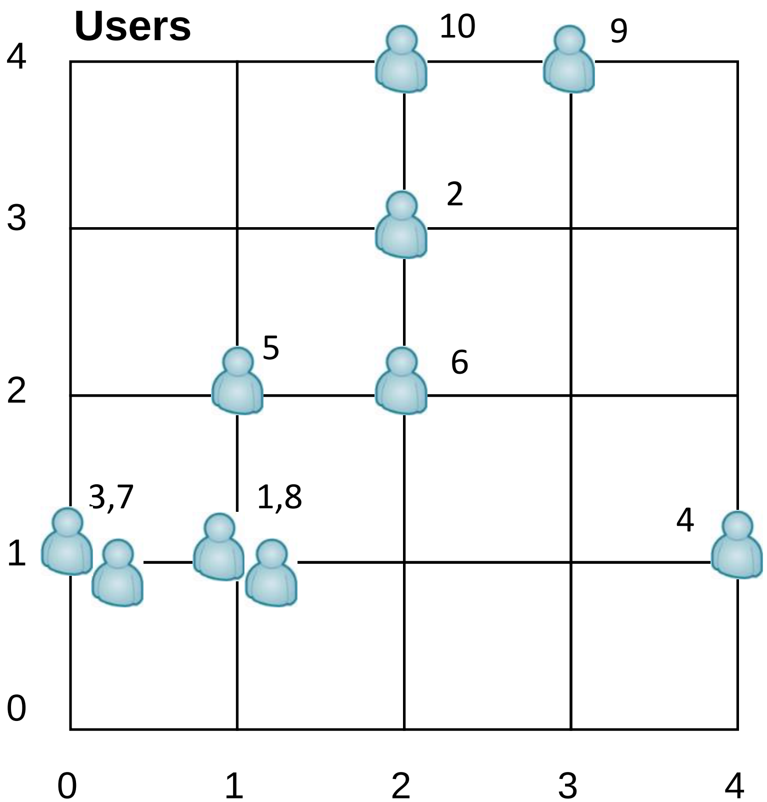
task: t4	105.8		
C(t4)	c1	c2	c3
Load if assignment			
c1	0.8233		
c2	0.7288	qmin	
c3	0.95	qmax	
RCL	q≤	0.8651	{c1,c2}

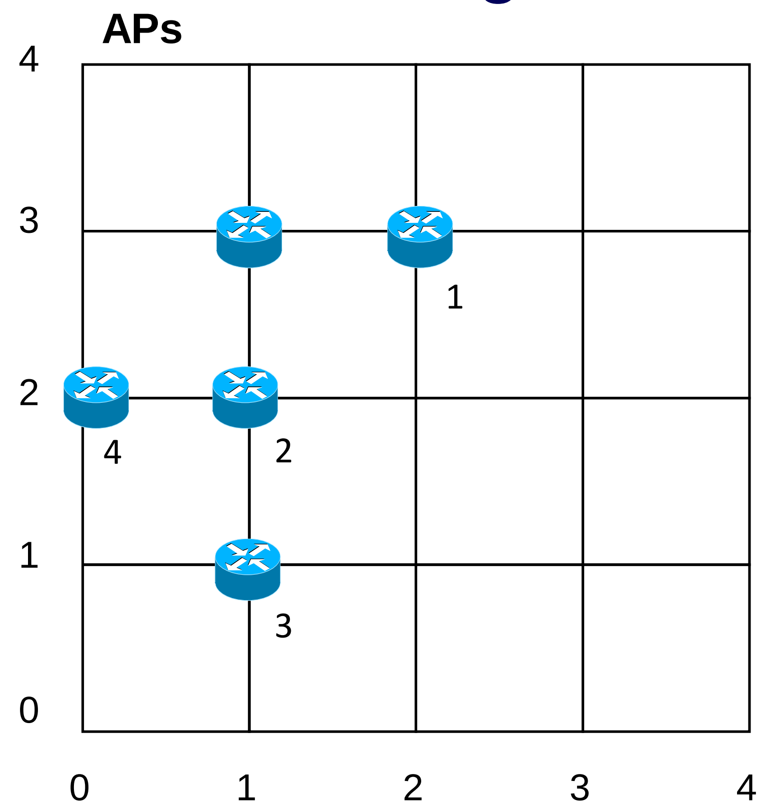
Computers	c1	c2	c3
residualCap	**89.4**	242.41	140.89
load	**0.823**	0.5187	0.799
S	{<t2,c3>,<t3,c1>,<t1,c2>,**<t4,c1>**}		

Solution

S	**{<t2,c3>,<t3,c1>,<t1,c2>,<t4,c1>}**
f(S)	**0.823**

Network Planning Problem





	R1	R2	R3
f	100	140	180
k	6	8	10
d	2	3	4

d(u,a)		u	1	2	3	4	5	6	7	8	9	10
		x	1	2	0	4	1	2	0	1	3	2
		y	1	3	1	1	2	2	1	1	4	4
1	2	3	2.2	0.0	2.8	2.8	1.4	1.0	2.8	2.2	1.4	1.0
2	1	2	1.0	1.4	1.4	3.2	0.0	1.0	1.4	1.0	2.8	2.2
3	1	1	0.0	2.2	1.0	3.0	1.0	1.4	1.0	0.0	3.6	3.2
4	0	2	1.4	2.2	1.0	4.1	1.0	2.0	1.0	1.4	3.6	2.8
5	1	3	2.0	1.0	2.2	3.6	1.0	1.4	2.2	2.0	2.2	1.4
a	x	y										

α=0.5

Network planning: Iterative execution (1/5)

#1

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)	100	100	100	140	100	100	100	100	100	100
a	3	1	3	1	2	1	3	3	1	1
d(u,a)	0.0	0.0	1.0	2.8	0.0	1.0	1.0	0.0	1.4	1.0

qmin=100, qmax=140, q<=120

a	1	2	3	4	5
m			R1		
U(a)			{3}		
km-cr			2		

#2

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)	0	40		40	0	0	0	0	80	80
a	3	3	3	3	3	3	3	3	3	3
d(u,a)	0.0	2.2	1.0	3.0	1.0	1.4	1.0	0.0	3.6	3.2

qmin=0, qmax=80, q<=40

a	1	2	3	4	5
m			R2		
U(a)			{3,4}		
km-cr			3		

Network planning: Iterative execution (2/5)

#3

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)	0	0			0	0	0	0	40	40
a	3	3	3	3	3	3	3	3	3	3
d(u,a)	0.0	2.2	1.0	3.0	1.0	1.4	1.0	0.0	3.6	3.2

qmin=0, qmax=40, q<=20

a	1	2	3	4	5
m			R2		
U(a)			{1,3,4}		
km-cr			1		

#4

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)		40			40	40	0	40	100	100
a	3	3	3	3	3	3	3	3	1	1
d(u,a)	0.0	2.2	1.0	3.0	1.0	1.4	1.0	0.0	1.4	1.0

qmin=0, qmax=100, q<=50

a	1	2	3	4	5
m			R2		
U(a)			{1,3,4,7}		
km-cr			0		

Network planning: Iterative execution (3/5)

#5

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)		100			40	40		40	100	100
a	3	1	3	3	3	3	3	3	1	1
d(u,a)	0.0	0.0	1.0	3.0	1.0	1.4	1.0	0.0	1.4	1.0

qmin=40, qmax=100, q<=70

a	1	2	3	4	5
m			R3		
U(a)			{1,3,4,6,7}		
km-cr			0		

#6

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)		100			100			100	100	100
a	3	1	3	3	2	3	3	2	1	1
d(u,a)	0.0	0.0	1.0	3.0	0.0	1.4	1.0	1.0	1.4	1.0

qmin=100, qmax=100, q<=100

a	1	2	3	4	5
m	R1		R3		
U(a)	{9}		{1,3,4,6,7}		
km-cr	3		0		

Network planning: Iterative execution (4/5)

#7

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)		0			0			0		40
a	3	1	3	3	1	3	3	1	1	1
d(u,a)	0.0	0.0	1.0	3.0	1.4	1.4	1.0	2.2	1.4	1.0

qmin=0, qmax=40, q<=20

a	1	2	3	4	5
m	R1		R3		
U(a)	{2,9}		{1,3,4,6,7}		
km-cr	0		0		

#8

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)					40			40		80
a	3	1	3	3	1	3	3	1	1	1
d(u,a)	0.0	0.0	1.0	3.0	1.4	1.4	1.0	2.2	1.4	1.0

qmin=40, qmax=80, q<=60

a	1	2	3	4	5
m	R2		R3		
U(a)	{2,5,9}		{1,3,4,6,7}		
km-cr	0		0		

Network planning: Iterative execution (5/5)

#9

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)								40		100
a	3	1	3	3	1	3	3	1	1	5
d(u,a)	0.0	0.0	1.0	3.0	1.4	1.4	1.0	2.2	1.4	1.4

qmin=40, qmax=100, q<=70

a	1	2	3	4	5
m	R3		R3		
U(a)	{2,5,8,9}		{1,3,4,6,7}		
km-cr	0		0		

#10

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4
q(u)										100
a	3	1	3	3	1	3	3	1	1	5
d(u,a)	0.0	0.0	1.0	3.0	1.4	1.4	1.0	2.2	1.4	1.4

qmin=100, qmax=100, q<=100

a	1	2	3	4	5
m	R3		R3		R1
U(a)	{2,5,8,9}		{1,3,4,6,7}		{10}
km-cr	0		0		2

Solution Cost=460

Parameter tuning

RCL (**min** greedy cost)

9	**10**	**11**	**11**	**13**	**18**

$\boldsymbol{\alpha_{min}}$**=0.3**

$q_{min}=9$
$q_{max}=18$
$q(e) \leq 9+\alpha_{min}(18-9)=11.7$

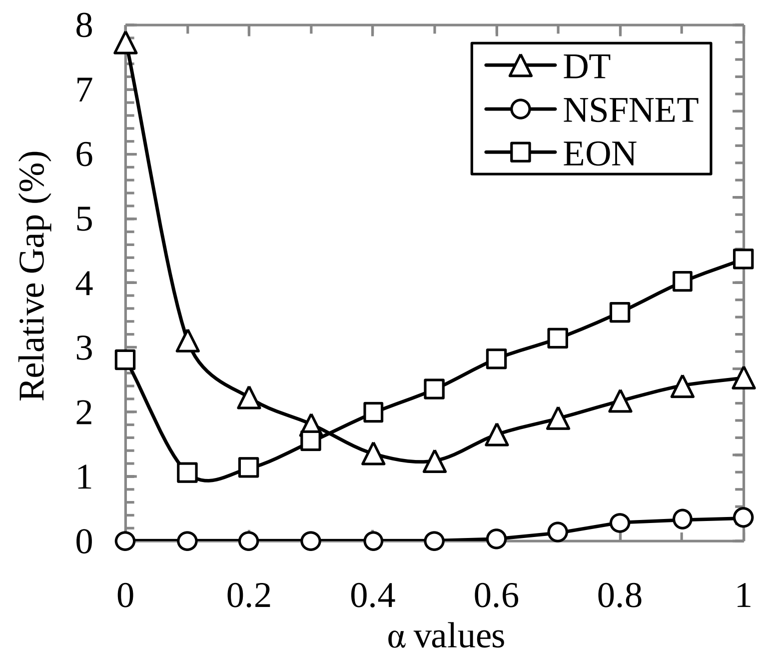
RCL (**max** greedy cost)

18	**13**	**11**	**11**	**10**	**9**

$\boldsymbol{\alpha_{max}}$ **=0.7**

$q_{min}=9$
$q_{max}=18$
$q(e) \geq 18-\alpha_{max}(18-9)=11.7$

To decide the value of α that fits the best for our optimization problem, we have to solve one or more *test instances* using different values of α



Stop Criteria

- Different stop criteria can be devised:
  - Execution time
  - Number of iterations
  - Goodness of the solution (value of the incumbent w.r.t. a given value)
  - Time since last incumbent update
  - Iterations since last incumbent update
  - A mix of the above

$S_{best} \leftarrow \{\}$

Stop Criteria

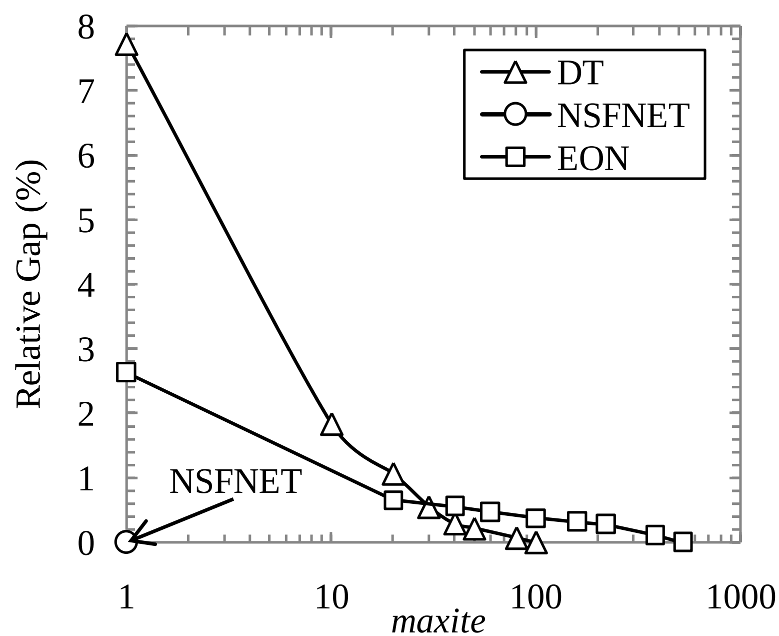
for $\boxed{k\text{=1..Max\_Iterations}}$ **do**

S = doConstructionPhase ()

S = doLocalSearch (S)

if f(S) < f(S_{best}) **then** $S_{best} \leftarrow S$

return S_{best}



Algorithmic Methods for Mathematical Models (AMMM)

Greedy Randomized Adaptive Search Procedure (GRASP)

Luis Velasco

(lvelasco @ ac.upc.edu)

Campus Nord D6-107