Algorithmic Methods for Mathematical Models (AMMM)

Local Search

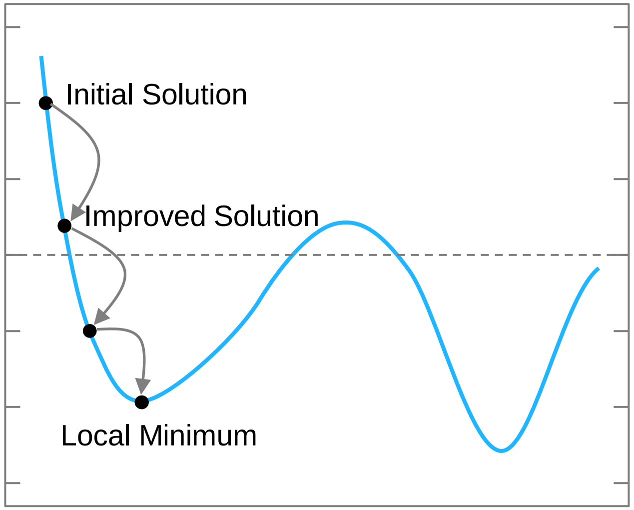
Luis Velasco

(lvelasco @ ac.upc.edu)

Campus Nord D6-107

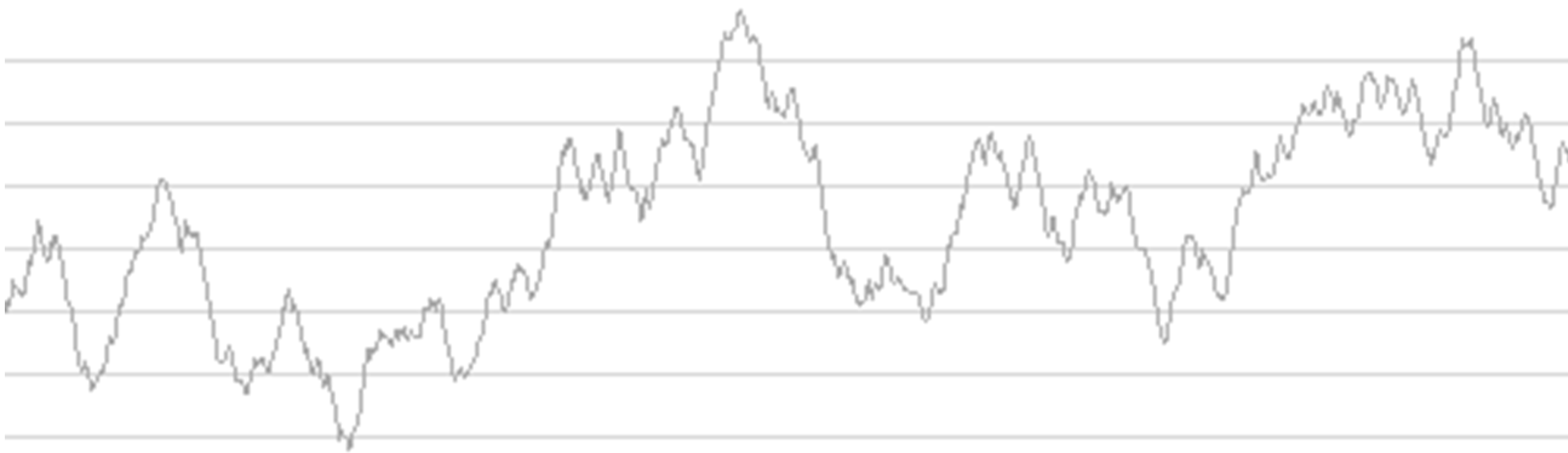
Local Search (LS)

- LS is as an iterative search procedure that, starting from an initial feasible solution, progressively improves it by applying a series of local modifications (or **moves**).
- At each iteration, the search moves to an improving feasible solution that **differs only slightly** from the current one.
- The search terminates when it encounters a **local optimum** with respect to the transformations that it considers.



Quality of the solutions

- Unless one is extremely lucky, this local optimum is often a **fairly mediocre solution**.

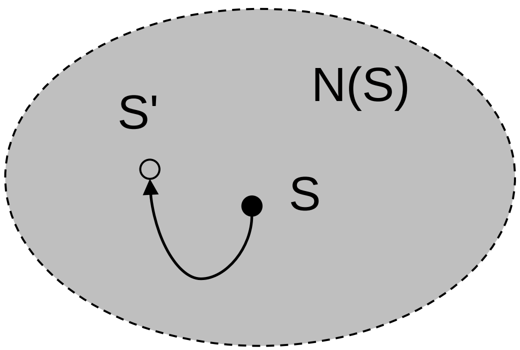


- The quality of the solution and computing times are usually **highly dependent** upon the “richness” of the set of moves considered at each iteration.

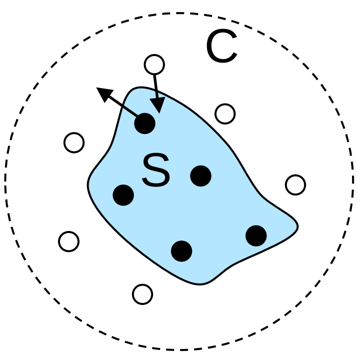
Algorithm and Search Strategies

Given a solution *S*
while *S* is not locally optimal **do**
    Find *S'* in N(*S*) s.t. $f(S') < f(S)$
    $S \leftarrow S'$
return *S*

- N(S) is the set of solutions that can be created by **exchanging** elements in the solution and elements not in the solution.

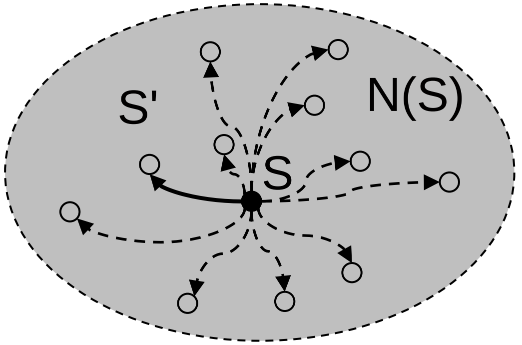


- A new solution is then created by removing from the solution a set *A* of elements in *S* and adding to the solution another set *B* not in *S*, where |A|>0 and |B|≥0.
- Examples:
  - **Removing one** element from S (|*A*|=1 and |*B*|=0)
  - **Removing by exchanging two** elements in S **and adding one** element in C\S (|*A*|=2 and |*B*|=1).
  - **Exchanging one** element in S and adding one element in C\S (|*A*|=1 and |*B*|=1).
- Simple neighborhoods (N(*S*)) are usually defined.

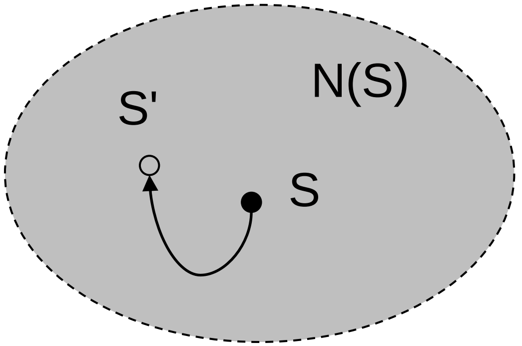


Search Strategies

- The neighborhood search may be implemented using either a *best-improving* or a *first-improving* strategy.
  - **Best-improving** strategy: **all neighbors** are investigated, and the current solution is replaced by the best neighbor.

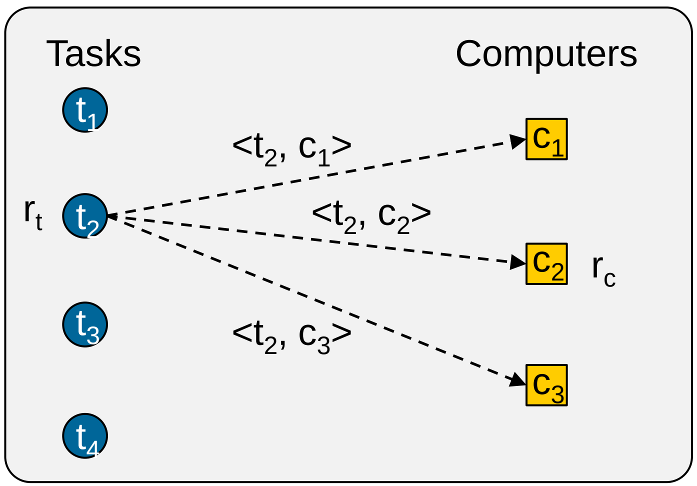


  - **First-improving** strategy: the current solution moves to the **first neighbor** whose cost function value is smaller than that of the current solution.

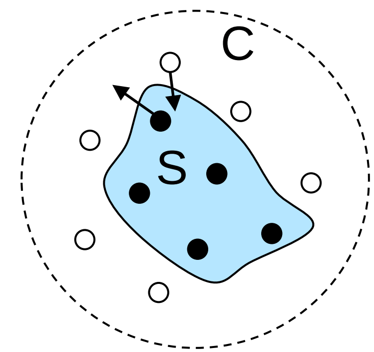


Neighborhoods

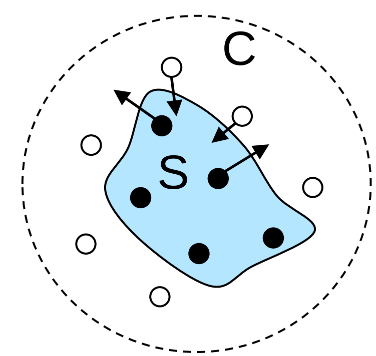
Tasks to Computers Assignment



- **Reassignment neighborhood**: reassignment of a task from a computer to another (**one-element exchange**).
  - Advantages: Easy and flexible: O(|*T*|*|*C*|)
  - Weaknesses: Limited improvement is obtained when computers are very busy.

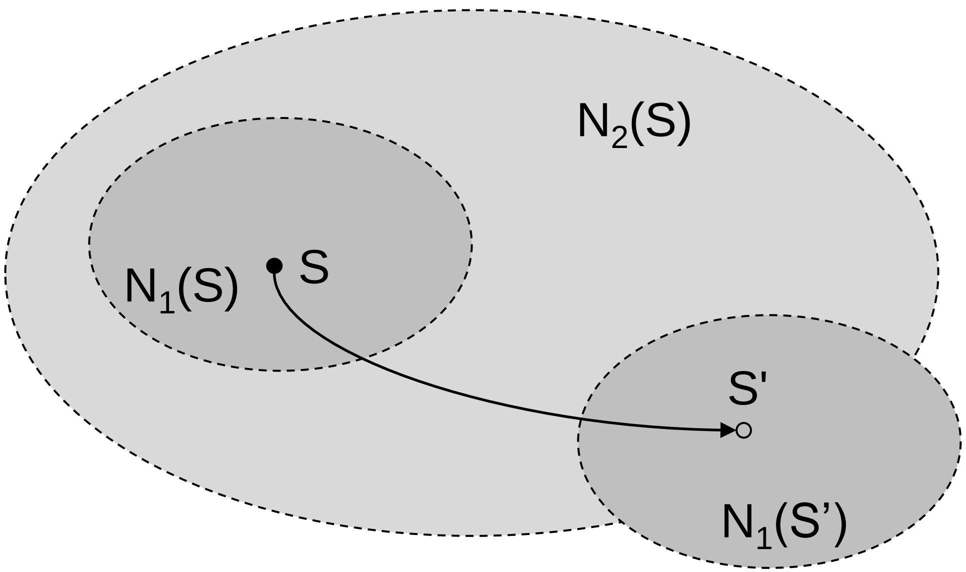


- **Assignment exchange neighborhood**: reassignment of tasks between computers (k=1). (**two-element exchange**).
  - Advantages: the number of feasible exchanges is increased
  - Weaknesses: higher complexity O(|*T*|^2) and maintains the structure of the solution from the one obtained with a greedy.

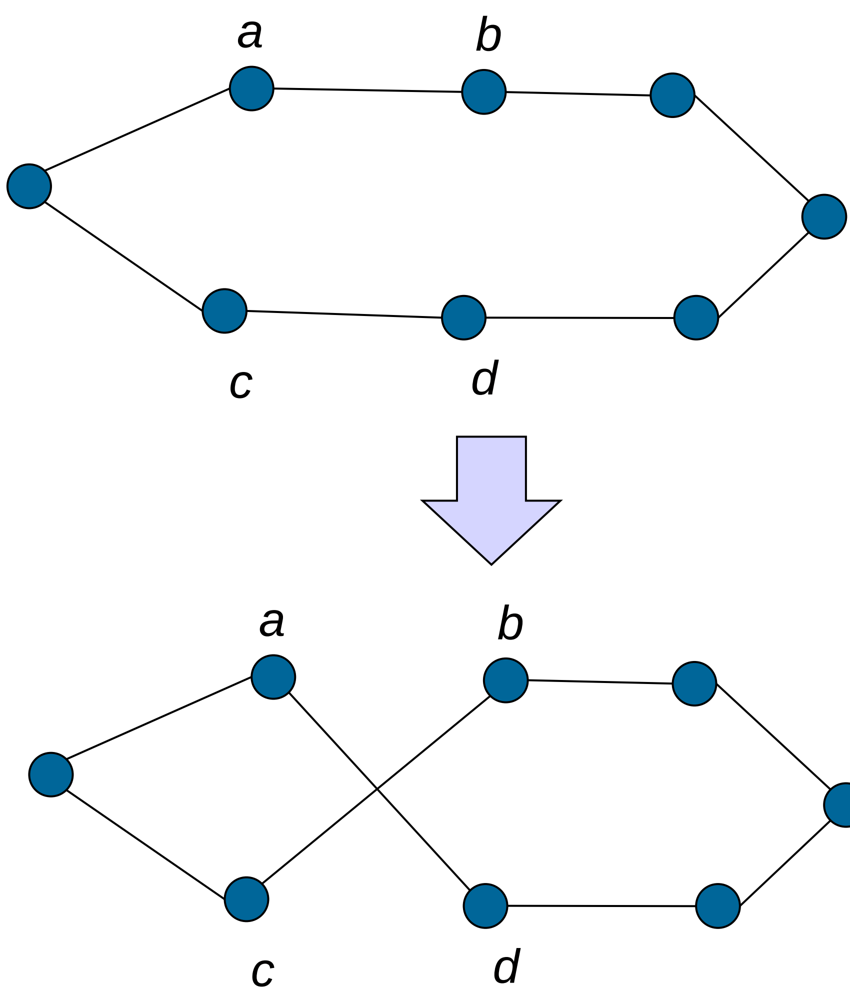


Variable Depth Search

- There is no one perfect neighborhood => multiple neighborhoods need to be explored.
- A set of neighborhoods with increasing complexity can be defined (VDS)



Example: TSP



Lin–Kernighan is one of the best heuristics for solving the TSP.
It involves swapping pairs of sub-tours to make a new tour.

Exchange non successive edges, if

$$d(a,b) + d(c,d) > d(a,d) + d(b,c)$$

N(S) = {S' created by exchanging 2 edges}

2 edges leave *S* and 2 edges enter into *S'*.

Lin–Kernighan is adaptive and at each step decides how many links between cities need to be switched to find a shorter tour. => **Variable neighborhoods**

Example: 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

Optimal solutions (cost 5)
S={p6, p7}
S={p2, p3, p6}

Greedy Solution S={p4,p6} → Cost: 6

We could try to remove as many pj as possible from S. For example, if the solution would be S={p1, p4, p6}, we could remove p1 => **N_0(S)**
In addition, we can do exchanges => **N_1(S)**.

M	Times covered	Covered by
1	1	{p6}
2	1	{p4}
3	1	{p4}
4	2	{p4, p6}
5	1	{p6}

A good indicator would be to find how many times an element is covered and with which sets. We can exchange p4 by another subfamily pj such that it cover only elements covered by p4 only once. In this case elements {2,3}.

p7 can be chosen to enter the solution and obtain a new solution:
S={p6, p7} → Cost: 5

Example: Assign tasks to computers (lab session 2)

while True

$sortedC \leftarrow$ sort $S.C$ by computers' load (DESC)

$moves \leftarrow \{\}$

for i=1..|$S.C$|-1 **do**

$c \leftarrow sortedC[i]$

for each t in $T(c)$ **do**

for j=i+1..|$S.C$| **do**

$c' \leftarrow sortedC[j]$

for each t' in $T(c')$ **do**

if $(r_{t'} - r_t) \leq residualCapacity(c)$ **AND** $(r_t - r_{t'}) \leq residualCapacity(c')$ **then**

$minResidualCapacity \leftarrow \min\{residualCapacity(c) + r_t - r_{t'},$
$residualCapacity(c') + r_{t'} - r_t\}$

$improvement \leftarrow minResidualCapacity - residualCapacity(c)$

if $improvement > 0$ **then** $moves \leftarrow moves \cup \{<t,c,t',c', improvement>\}$

if $moves=\{\}$ **then return** S

$move=<t,c,t',c',improvement> \leftarrow \text{argmax}\{improvement \mid move \text{ in } moves\}$

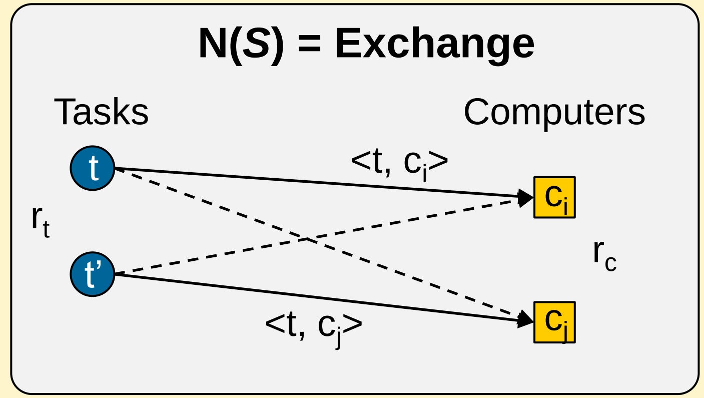
$T(c) \leftarrow T(c) \setminus \{t\} \cup \{t'\}$

$T(c') \leftarrow T(c') \setminus \{t'\} \cup \{t\}$

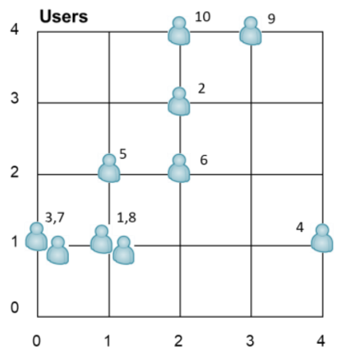
$residualCapacity(c) \leftarrow residualCapacity(c) + r_t - r_{t'}$

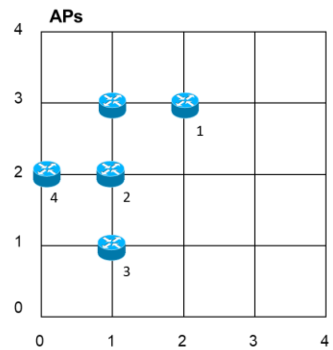
$residualCapacity(c') \leftarrow residualCapacity(c') + r_{t'} - r_t$

We could also define the improvement related to that of the highest loaded computer (objective function). Pros and const.



Example: Network planning





***N*(S) = reassignment**

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

Given Solution *S*

sort *A* by router cost

for *a*=1..|*A*|-1 **do**

  for each *u* in *U*(*a*) **do**

    for *a'*=*a*+1..|*A*| **do**

      if <*u*, *a'*> is feasible and f(S') ≤ f(S) **then**

        U(*a*) ← U(*a*)\{*u*}

        U(*a'*) ← U(*a'*) ∪ {*u*}

        recompute(*a*)

        recompute(*a'*)

        break

return *S*

Network planning: Local Minimum Solution

	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**										

u	1	2	3	4	5	6	7	8	9	10
cr	2	3	4	1	2	2	1	2	3	4

Greedy Solution Cost=460

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

Solution Cost=420

a	1	2	3	4	5
m	R3		**R2**	R1	
U(a)	{2,9,10}		{4,5,6,7,8}	{**1**,3}	
km-cr	0		0	0	

Algorithmic Methods for Mathematical Models (AMMM)

Local Search

Luis Velasco

(lvelasco @ ac.upc.edu)

Campus Nord D6-107