TABU SEARCH METHOD

1. HOW THE PROGRAM WORKS

- -Initialization
- -Structure of the algorithm
- -Evaluation Function
- -Successor Function

1.1- INITIALIZATION

- Mostly the usual: satellite data + Taboo Search data
- TabuList holds a cell per every station
- Integer representation of representative stations.

Not much difference between binary or integer.

```
%Binary Representation
selectedStations = [0 1 0 0 1 1];
%Integer Representation
selectedStations = [2 5 6];
%Binary to Integer switch
selectedStationsBin = zeros(1, length(stations));
selectedStationsBin(selectedStationsInt) = 1;
%Integer to Binary switch
selectedStationsInt = find(selectedStationsBin==1);
```

1.2- STRUCTURE OF THE ALGORITHM

- Very similar to the generic Taboo Search structure.
- Matrix of successors has two extra columns: the value of the successor, the change from the parent.
- Stopping conditions are a total maximum and a maximum without improvement: 500, 20.

Parent **1**	**2**	**5**	**Value** **138.42**	**Change** **-**
3	2	5	141.23	3
1	7	5	156.75	7
1	2	6	137.89	6

1.3- EVALUATION FUNCTION

- Standard method: get all the not-representative stations, then add the distances of each to its closest representative.

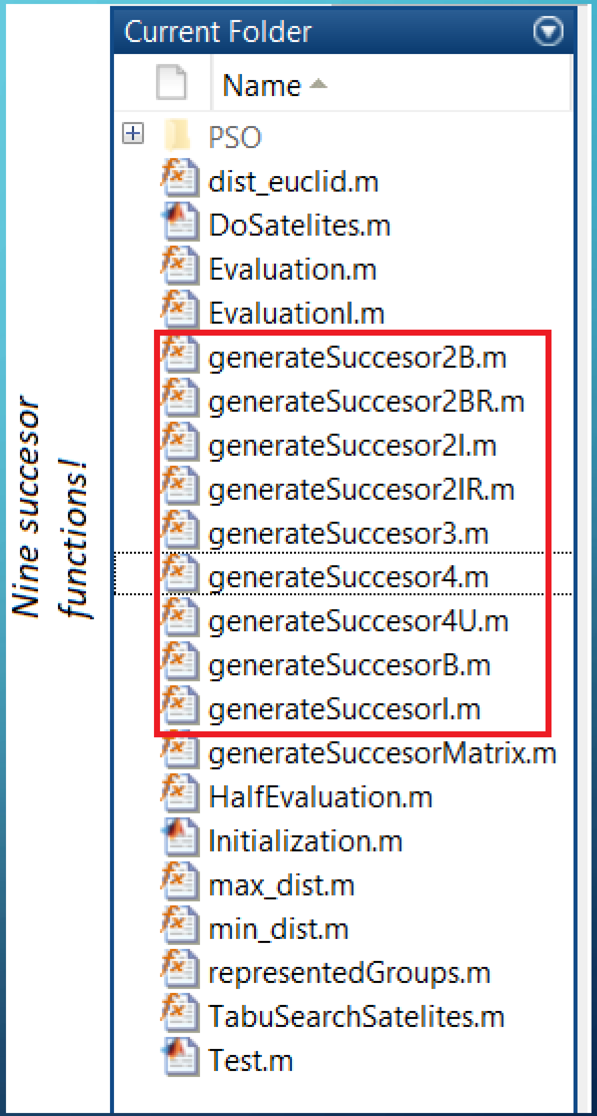
```
function value = EvaluationI(stations, selectedStations)
    value = 0;
    notReps = 1:length(stations);
    notReps(selectedStations) = [];
    for n = notReps
        value = value + min_dist(stations(:,n), stations(:, selectedStations));
    end
end
```

1.4- SUCCESSOR FUNCTION

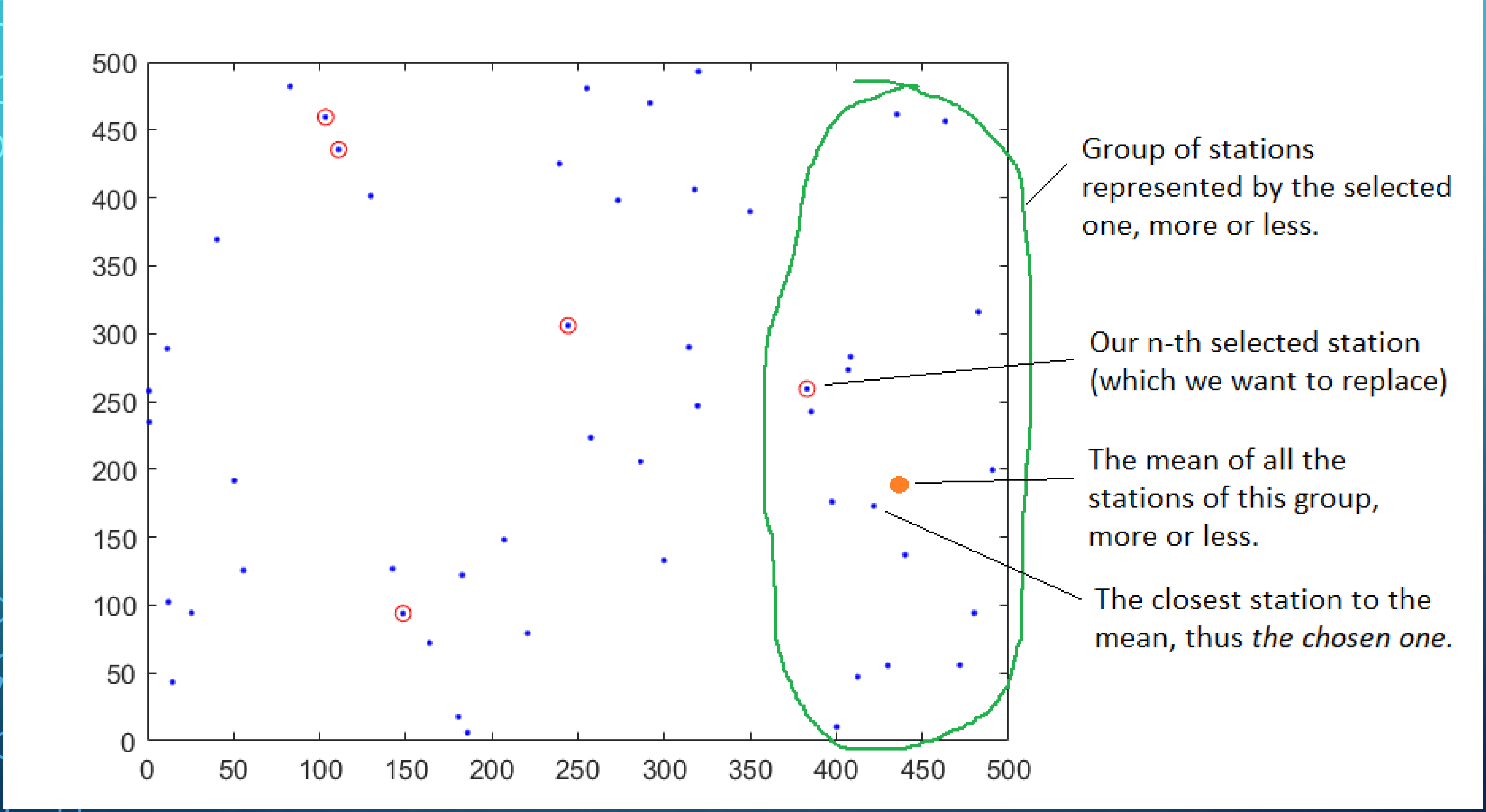
* Requires a varLevel parameter, determines the n-th selected station to be altered.

We tried different methods to replace the n-th selected:

- By the next one in the array
- By its closest station
- By the furthest station represented by it.
- By the station closest to the mean point of all of its group of represented stations.



In the end, the 4th method proved best, by a decent margin.



2. CHOICES OF IMPLEMENTATION

2.1- WHY INTEGER, NOT BINARY REPRESENTATION?

- In the end, both had extremely similar efficiencies when tested.

(Plus we can switch between them with ease)

- So, this didn't have a big impact on the overall efficiency.
- Integer used less memory, and it was easier to use with the following…

2.2- AUXILIAR ARRAY: "REPRESENTED"

- Length: N (500).
- In position n, stores the station that represents n.
- If n is a representative, stores a 0.

Representatives	**3**	**6**	**8**			
Represented	3	8	0	6	6	…

2.2.1- WHY IS THIS ARRAY USEFUL?

- Both the evaluation and the successor function use these “groups of represented”.
- Tests lead to the evaluation function being ~20 times faster with this array.
- Successor function went from 2*10e-3 to 4*10e-6.

500 times faster!!

2.2.2- THERE ARE DRAWBACKS…

- To create this auxiliar array we call the function: “representedGroups”.
- This function is very costly, order of N*M.
- A call to this function plus either evaluation or successor completely negates all improvement…

```
function reps = representedGroups(stations, selectedStations, N)
    reps = zeros(1, N);
    notReps = 1:N;
    notReps(selectedStations) = [];
Order N — for n = notReps
    Order M — [~,pos] = min_dist(stations(:,n), stations(:, selectedStations));
              reps(n) = selectedStations(pos);
          end
end
```

Order N*M

SOLUTION: MINIMIZE THE CALLS TO THE FUNCTION

```
    function sucsMatrix = generateSuccesorMatrix(stations, selectedStations, M)
        sucsMatrix = zeros(M, M+2);
        N = length(stations);
One call —— represented = representedGroups(stations, selectedStations, N);
        for varLevel = 1:M
M calls ——— sucsMatrix(varLevel, 1:M) = generateSuccesor4(stations, selectedStations, represented, varLevel)
            sucsMatrix(varLevel, M+1) = sucsMatrix(varLevel, varLevel);
            sucsMatrix(varLevel, M+2) = EvaluationI(stations, sucsMatrix(varLevel, 1:M));
        end
        sucsMatrix = sortrows(sucsMatrix, M+2);
    end
```

The successor function needs the represented array of the *father.*

1 father -> M successors.

We can't save the evaluation function, it stays inefficient :(

3.- CONCLUSIONS OF TABOO SEARCH

- It's relatively quick! We optimized every operation as much as possible.
- Total of iterations remain low. For 500 stations it finishes in under 100!
- Often gets good results: ~1.35 e4
- *Possible drawback? Results might become worse with low number of stations, due to the successor method.

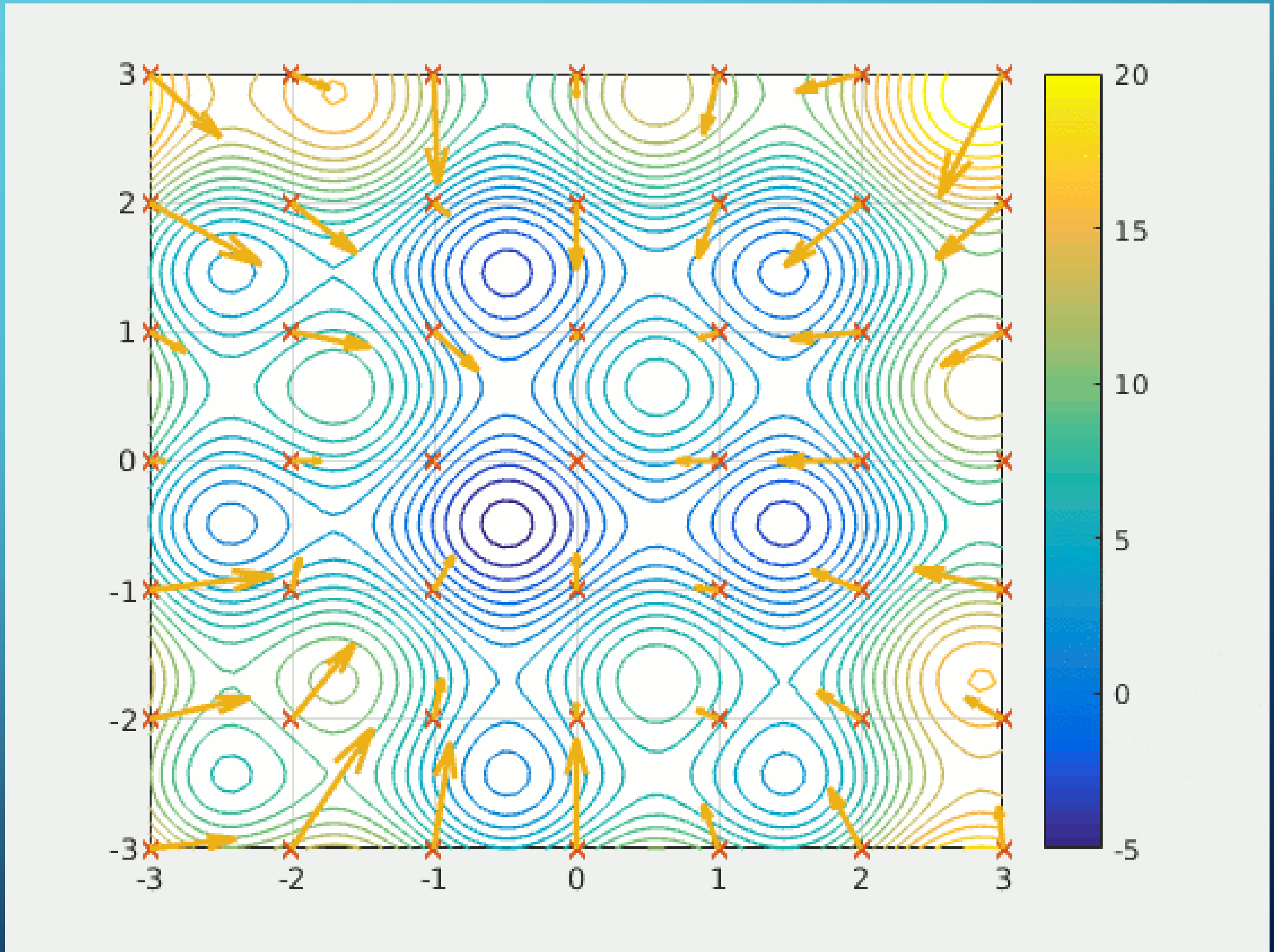
  But we couldn't really confirm it.

SATELLITES PROBLEM SOLUTION

SOLVED WITH PARTICLE SWARM OPTIMIZATION ALGORITHM

PSO - MOTIVATIONS

- Standard implementation
- Acceptable solutions
- Different approaches
- Convenient convergence



PSO – NOT ALL IS GOLD

- Local optimum not guaranteed
- Performance
- Best solution
- Tweaks needed

PSO – NOT ALL IS GOLD

- Local optimum not guaranteed
- Performance
- Best solution
- Tweaks needed

PSO - BASIC SCHEME - INITIALIZATION

- Similar to Taboo.
- Population settings provided
- “The keys”:
  - Speed
  - Position

```
% Pob setting
pobSize = 140;
pob = generatePob(N, pobSize, M);
% Represented data save
n = 1;
represented = zeros(pobSize, N);
while n <= pobSize
    represented(n,:) = representedGroups(stations, pob(n,:), N);
    n = n+1;
end
fitpob = EvalPob(pob, stations);
% initial particular best
particularBest = pob;
fitParticularBest = fitpob;
% 1º position: fitness, 2º position: index in pob
globalBest = [min(fitParticularBest) find(min(fitParticularBest) == fitParticularBest,1)];
% cell array to clasify speed
speed = cell(pobSize,3);
```

PSO – BASIC SCHEME – BODY ALGORITHM

- The same for any problem
- Deeper slight differences
- 18 functions in total

```
% Inertia applied over current speed
speed = Inertia(speed);
% computation of speed equation
speed = updateSpeed(speed, particularBest, globalBest, pob);
% speed sum
totalSpeed = speedSum(speed, pobSize);
% Position update
pob = updatePob(totalSpeed,pob);
fitpob = EvalPob(pob, stations);
[particularBest, fitParticularBest] =
updateFitPobBest(pob, particularBest, fitpob, fitParticularBest);
globalBest =
[min(fitParticularBest) find(min(fitParticularBest) == fitParticularBest,1)];
```

PSO – BASIC SCHEME - STOPPING CRITERIA

- Time
- Iterations
- Stallation
- Acceptable solution

```
while it <= 10 && totalIt <= 500 && globalBest(1) > 12300
```

SATELLITES - REPRESENTATION

- Floating satellites
- Random population
- Initial speed = 0

SATELLITES - REPRESENTATION

- Floating satellites
- Random population
- Initial speed = 0
- Inertia, particular and global coefficients previously decided

SATELLITES - REPRESENTATION

- Floating satellites
- Random population
- Initial speed = 0
- Inertia, particular and global coefficients previously decided
- Population size → "Big problems requires bigger populations"

SATELLITES - POSSIBILITIES

- Two of them:
  - Converge to the best computing differences

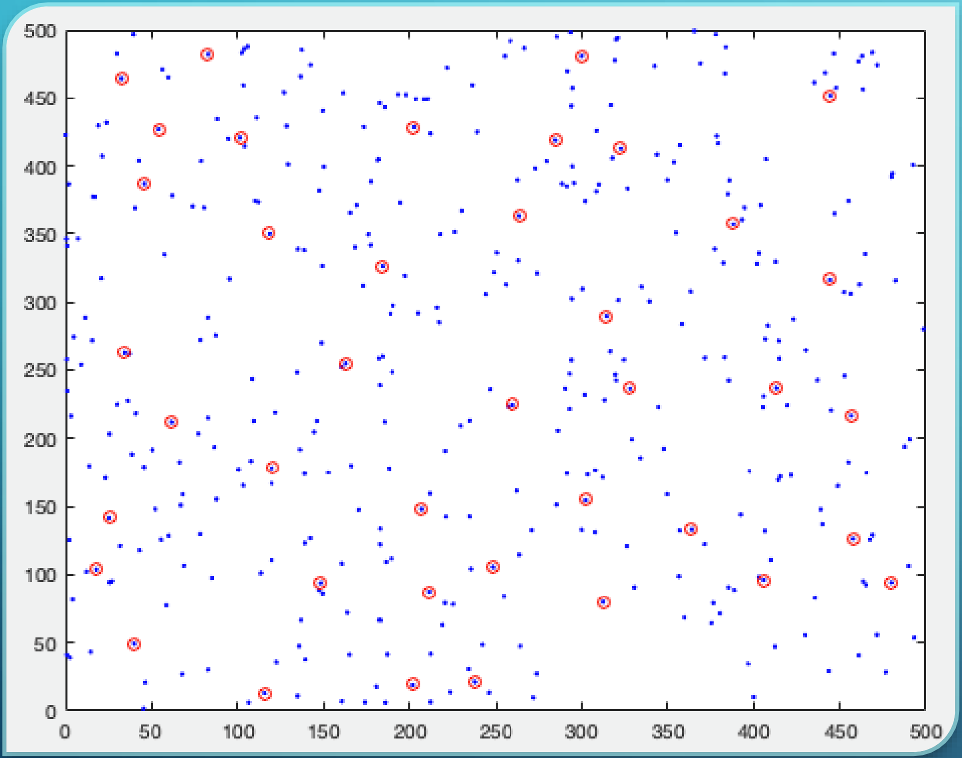
SATELLITES - POSSIBILITIES

- Two of them:
  - Converge to the best computing differences
  - **Converge to the best through exchanges**

PSO – OBTAINING RESULTS

- **BALANCE**

PSO – OBTAINING RESULTS



- **BALANCE**
- Testing

globalBest [1.1557e+04 31]

```
Time to compute problem measured in seconds:
   10.5105
```

THANK YOU ALL!

ANY QUESTIONS?