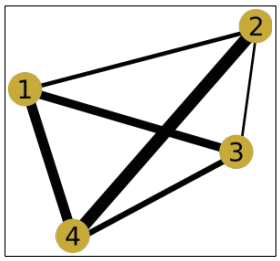
The Quadratic Assignment Problem

Martino Ferrari

1. Quadratic Assignment Problem

The quadratic assignment problem (QAP) is one of the fundamental combinatorial optimization problems.
It models one common logistic problem and it is part of the *facility location* category of problems.



The QAP models the following situation:
There are a set of n facilities and a set of n locations. For each pair of locations, a distance is specified and for each pair of facilities a weight is specified (e.g., the amount of supplies transported between the two facilities). The problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows.[1]

We can formalize the problem in this way:

$n \in \mathbb{N}$	the number of facilities
$D \in \mathbb{N}^n$	the distance matrix (symmetric with diagonal 0, as the distances are symmetric)
$W \in \mathbb{N}^n$	the weight matrix (symmetric with diagonal 0, as the flows are symmetric)

The space of solutions S is composed of all the possible ($n!$) permutations of the n facilities and the fitness is defined as $f(\psi) = \sum_{i,\, j=1..n} W_{[i][j]} \cdot D_{[\psi_i][\psi_j]}$, where ψ_i is the location of the facility i in the solution ψ.

2. Introduction to the Tabu Search metaheuristic

The Tabu Search (TS) metaheurisitic is an evolution of the *local search* heuristic created by Fred W. Glover in 1986.
The local search tries to find an acceptable solution to a problem by choosing the best neighbour of the currently selected solution. A limitation of this method is that it tends to get stuck in sub-optimal regions (where there are local minima or maxima) and so it limits the solution space (S) exploration.
To avoid this problem and explore regions otherwise left unexplored, the tabu search modifies the neighbourhood space $N(\psi_i)$ during the search.
To do so, it defines $N^1(\psi_i)$ as $\forall \psi' \in N^1(\psi_i) \rightarrow \psi' \in N(\psi_i) \wedge \nexists \psi' \in N^1(\psi) \rightarrow \psi' \in tl$.
The **tabu list** tl (the is not necessary implemented as a list) is a memory structure where, in the simplest case, the last l visited solutions (or movements done) are stored.
In this way, the search will walk away from the already visited solutions and it will explore more parts of S.
The size of l (tabu tenure) is very important and can modifiy the algorithm behaviour (I will explain and show it later).

1 From QAP wikipedia page

3. Tabu Search and the QAP

To implement any metaheuristic there is the need to define 5 elements:

- The search space S
- The start solution
- The neighbourhood of a solution $N(\psi)$
- The exploration operator U
- The stop condition

We already defined $S \in \mathbb{R}^n$ as the set of all possible permutations of n facilities.

The **start solution** will be generated randomly.

The **neighbourhood** $N(\psi)$ of a solution ψ is composed of all the solutions ψ' with only two facilities swapped from the original one.

In this way the neighbourhood size will be $n \cdot (n-1)/2$ but, as said before, the tabu search modifies the neighbour set during the research, by using the tabu list. In this case the elements of the tabu list are represented as pairs (i, r), being i a facility and r a location.

This means that a neighbour swapping the facilities i, j in the respective locations s, r is forbidden if both (i, r) and (j, s) are in the tabu list, meaning that the same permutation was tested in the last l iterations.

The U operator will choose the best (in terms of fitness, in this case the lowest fitness) neighbour in the set $N^1(\psi)$. To improve the performance we implement both an **aspiration** and **diversification** mechanism.

With the aspiration mechanism U will choose a neighbour ψ' even if it is in the tabu list if its fitness is better of the best found one.

With the diversification mechanism U will choose a neighbour ψ' even not having the best fitness (between the neighbours) if the associated permutation was not chosen in the last u iterations (with in our case $u = n^2$).

Finally the **stop condition** is defined as a the number of iterations to do.

4. Implementation

The peculiarities of this implementation are 3:

- The use of a $n \times n$ matrix as tabu list
- The use of the same matrix for the diversification process
- The use of Δfitness to speed up the computation of the neighbours fitness

The tabu matrix (**tm**) represents all the n possible facilities and all the n possible locations.

When moving from a solution to the next one, by swapping the facility i at the location r with the facility j at the location s, at the iteration t the algorithm stores t in the matrix at the (i, r) and (j, s) positions. A permutation is forbidden simply if $t' \leq (tm[i', s'] + l)$ and $t' \leq (tm[j', r'] + l)$ where l is the tabu tenure.

For the diversification mechanism we know that we have to perform a permutation if it has not been done in the last u iterations. We verify this condition, at the iteration t'. by simply checking that $t' \leq (tm[i', s'] + l)$ and $t' > (tm[j', r'] + u)$.

The definition of the fitness of a solution in the QAP is:

$$f(\psi)=\sum_{i,\,j=1..n} W_{[i][j]}\cdot D_{[\psi_i][\psi_j]}$$

meaning that the time to compute one-solution fitness is n^2. Being as the D and W matrices are symmetric and with diagonal 0 (the distance between a and b or b and is the same and the distance between a and a is 0) the time can be reduced to $n\cdot(n-1)/2$ (note: still of order $O(n^2)$).
The number of neighbours of a solution is, as well, $n\cdot(n-1)/2$, meaning that the complexity to compute all the the neighbourhood fitness would be of order n^4.
To quicker compute a neighbour the fitness is possible to use the partial cost, or **Δfitness.** In fact the difference of the fitness between a solution ψ and one of its neighbour depends only on the (i,j) switched facilities and can be computed as:

$$\Delta f(\psi,i,j)=2\cdot\sum_{k\neq i,\,j}^{n}(W_{[i][k]}-W_{[j][k]})\cdot(D_{[\psi_j][\psi_k]}-D_{[\psi_i][\psi_k]})$$

In this way, the complexity to compute the fitness of the neighbourhood would be of type n^3.
Finally, I implemented the algorithm both in **python** and **C++** for two reasons, first to verify my results by rewriting the algorithms from scratch.,and second to improve the performance (over 50 times better with no optimization in the code).

5. Results

Before starting to analyze the performance and the results of the metaheuristic, I tried to better understand the problem landscape. If not specified, the analysis is done using the "1.dat" file representing a QAP of size 12. I choose to explore S by generating 1000 random solutions and computing their fitness.
The result was the following: $\mu:813$, $\sigma:40$

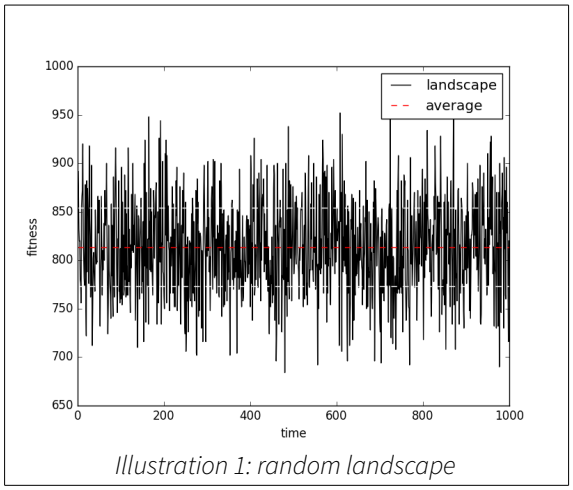


Illustration 1: random landscape

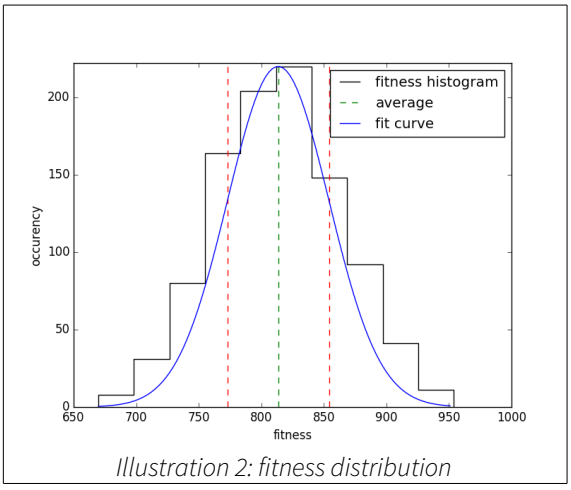


Illustration 2: fitness distribution

Then running the metaheurisitic over the same file multiple times gave me a stable optimal solution of **578**, using the parameters found before, the probability of this results is around 3.2×10^{-8}.
As the size of S is $12!\simeq5^8$ and it's still a reasonable number, I choose to perform a brute force research in the solution space to look for any better solution. Few minutes of computation showed that **578** is not just a local

minimum but the **global** one (actually there are around 15 solutions with fitness of 578). The fact that 578 is a global minimum and that the size of S is reasonably small explains the stability of the solution using the metaheuristic.

As pointed before the metaheurisitic behaviour is influenced by two factors, the size of l and the optional diversification process.
To better understand how the search behaves, I choose to plot the fitness of the explored solutions versus time for different configurations (always using the file "1.dat" as source)

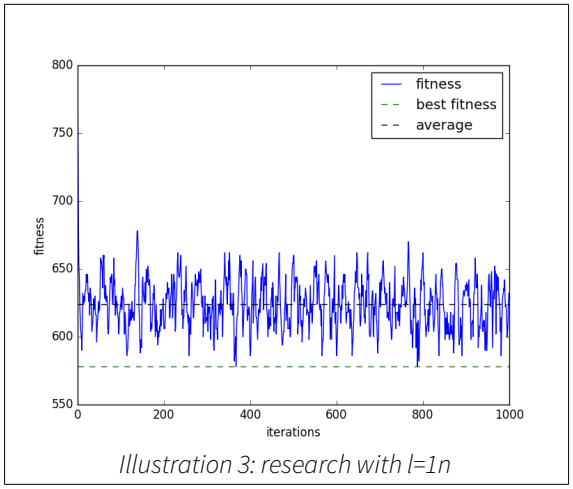


Illustration 3: research with l=1n

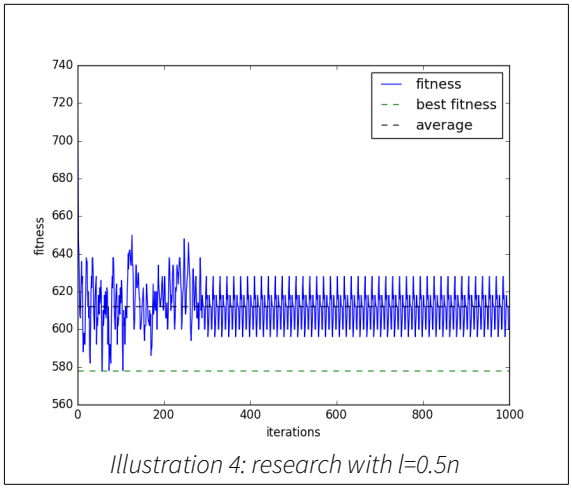


Illustration 4: research with l=0.5n

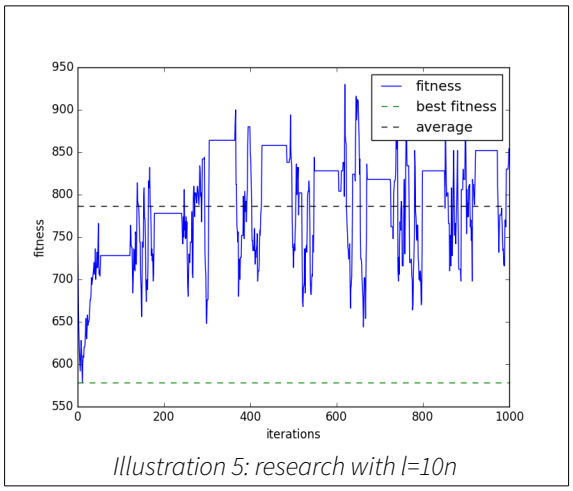


Illustration 5: research with l=10n

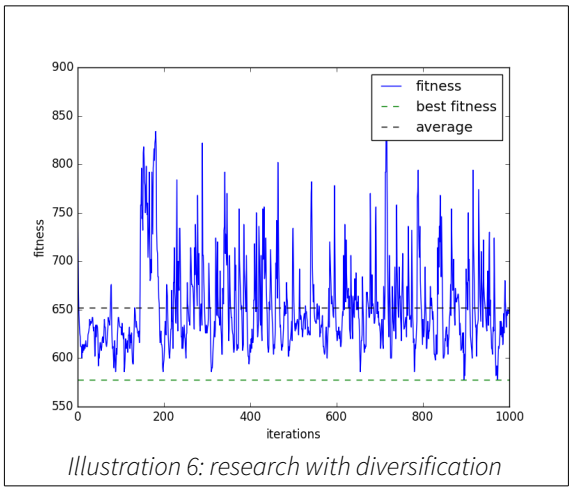


Illustration 6: research with diversification

In illustration 3 the configuration was $l=n$ and no diversification, it resulting in a research without any pattern repetition (meaning the research is not trapped in any local/global minimum) with mean fitness around 640, reached after few iterations.
With a much smaller ratio l/n, like in illustration 4, for wich $l=0.5\cdot n$, it's possible to see that the algorithm gets stuck in some local minimum, because the tabu list empties itself too quickly and the algorithm falls back. With lower l the algorithm is privileging the intensification, that's why the average fitness in this case is around 610, instead of 640 before.
When otherwise the ratio l/n is way too big, like in illustration 5 where $l=10\cdot n$, the algorithm will end up in situations where all the neighbours are forbidden, and (in my case) it will end up skipping many iterations

waiting to have at least one free neighbour. More over is possible to see that with higher l the algorithm is diversifying much more, looking in solutions with worst fitness to explore new areas. In fact in this case the average fitness it's around 790 (very close to the landscape average) .

Finally it's possible to have a good mixture of diversification and intensification by using the diversification process (in the plot is easy to see that starts after n^2 or 144 iterations). In this case the algorithm will intensify the research in never explored areas(instead of only diversifying), as easily recognizable in illustration 6, where the average still around 650 but it's possible to see that the algorithms searched in different unexplored areas (where the high fitness peaks are).

In the first experiment I will try to vary l and enable or disable the diversification.

Experiment over file 1.dat			
	L=1n	L=0.9n	L=0.5n
Diversification off	Bf=578, μ=578, σ=0	Bf=578, μ=580, σ=3	Bf=578, μ=584, σ=3.5
Diversification on	Bf=578, μ=578, σ=0	Bf=578, μ=578, σ=0	Bf=578, μ=578, σ=0

The results are, as expected, better for values of l close to n or with diversification on, for the reasons already explicated before.

In the next experiment I will execute the metaheurisitic over QAP with different sizes but same settings, $l=0.5n$ and diversification enabled, to have a good mixture of diversification and intensification .

Experiment over QAP with different sizes (with diversification on)					
N	File	Iterations	Best fitness	Mean	Standard deviation
40	40.dat	20000	491,718	493,374	1,033
50	50.dat	20000	1,259,690	1,264,160	1,988
80	80.dat	20000	8,664,514	8,695,340	12,404
100	100.dat	5000	21,756,548	21,803,300	22,513

7. Project structure

cpp/	contains the c++ implementation of the metaheuristic used for the experiments
cpp/bin	contains the compilation script and the *.dat file generated for the experiments
→ analyser	performs analysis over one run of n iterations of the metaheurisitic
→ generator	generates a dat file containing the n,D,W matrix of a random QAP problem of size n
→ statistic	performs analysis over multiple executions of the metaheurisitic over the same QAP problem
→ landscape	performs some analysis over the landscape of a QAP problem
→ brute_force	performs a brute force search of a QAP problem
python/	contains the python code used for the first implementation of the metaheurisitic
README	contains the instruction to compile the codes