Report for the project for the Swarm Intelligence course

Student **Aldar Saranov (000435170 ULB)**
Aldar.Saranov@ulb.ac.be

Tuesday 5th June, 2018

Contents

1 Program implementation

In the literature many different methods are proposed and researched including various implementations of the simulated annealing, taboo search, hybrid genetic-taboo search. However, for this project an algorithm known as Hybrid Ant System for the Quadratic Assignment Problem (HAS-QAP) was used as it was proposed by Gambardella and Dorigo in [**?**]. As all the ACO algorithms it uses the notion of solution construction biasing by means of pheromone trails, deposited by ants. The high-level outline of HAS-QAP is shown in Figure 1. It was implemented in two versions - with Rank-based Ant System (RAS) and Elitist Ant System (EAS) as different pheromone trail update techniques.

Let n be the number of facilities/locations, i.e. the size of a problem. Every solution of a QAP problem is a permutation ψ of an integer sequence from 1 to n.

The HAS-QAP includes such components as:

- Random solution generation
- Local Search
- Pheromone trail swaps
- Intensification
- Pheromone update
- Diversification

Listing 1: HAS-QAP pseudo-code

```
1  procedure HAS–QAP
2  generate m random permutations ψ^1 ,... ,ψ^m .
3  [optional] improve ψ^1 ,... ,ψ^m by local search
4  let pi* be the best solution
5  initialize the pheromone trail matrix T
6  activate intensification
7
8  while (there is time left)
9    for k from 1 to m
10     ψ̂^k = PheromoneTrailSwaps(ψ^k)
11     [optional] improve ψ̂^k by local search to get ψ̃^k
12   end
13
14   for k from 1 to m
15     if intensification is active then
16       ψ^k = best(ψ^k;ψ̃^k)
17       if none of ψ^k changed then
18         disable intensification
19     else
20       ψ^k = ψ̃^k
21   end
22
23   if exists ψ̃^k better then ψ*
24     update the new best ψ* = ψ̃^k
25     activate intensification
26   end
27
28   update the pheromone trail matrix
29
30   if S iterations in a row are not improving then
31     perform diversification
32 end
```

Some micro-optimization were applied to the original version of HAS-QAP such as reorganizing conditional branches. For example, we extracted the conditional block on the line 15 outside the loop to avoid redundant condition checks.

1.1 Random solution

Is used in the initializing section of the algorithm. Generate m random solutions. In out implementation, the algorithm takes the facilities one by one and assigns it to one of the free locations according to random uniform rule. This is an exploration step.

1.2 Local Search

We implemented the same local search that is described in the paper. The implemented local search is based on sequential random check of all pairs i and j and performing swaps of location between the i-th and j-th facilities, in case if these swaps are profitable. For this we compute the difference of objective values Δ before the swap and after. Instead of full objective value recomputation in $O(n^2)$, one can compute Δ value in an optimized $O(n)$ way as in Formula 1.

$$\Delta(\psi, i, j) = (b_{ij} - b_{ji}) \times (a_{\pi_i\pi_j} - a_{\pi_j\pi_i}) + \sum_{k=1}^{n} [b_{ik} \times (a_{\pi_i\pi_k} - a_{\pi_j\pi_k}) + b_{ki} \times (a_{\pi_k\pi_i} - a_{\pi_k\pi_j}) + b_{jk} \times (a_{\pi_j\pi_k} - a_{\pi_i\pi_k}) + b_{kj} \times (a_{\pi_k\pi_j} - a_{\pi_k\pi_i})] \quad (1)$$

Listing 2: Local Search pseudo-code

```
1  procedure LocalSearch(solution ψ)
2  I = ∅
3  while (|I| < n)
4     pick i uniformly randomly, i ∉ I
5     J = {i}
6     while (|J| < n)
7        pick j uniformly randomly, j ∉ J
8        if (Δ(ψ, i, j) < 0)
9           exchange ψ_i and ψ_j
10          J = J ∪ {j}
11       end
12       I = I ∪ {i}
13    end
14 end
```

Thus, this local search allows only improving moves and leads to high intensification. The total local search complexity is $O(n^3)$.

1.3 Pheromone trail swaps

Pheromone trail value τ_{ij} is assigned to every pair of facility i and location j. The more this value is, the more strongly the algorithm will try to bias to assigning the facility i to the location j.

Pheromone trail swaps are applied on each iteration for each solution. These swaps have two policies - exploring and exploiting. For a given solution, an exploiting policy is applied with probability q. This parameter is the key parameter that defines the trade-off between exploiting and exploring.

In exploiting policy, a random facility r is chosen. Then a facility s is chosen, in such way, that the value $\tau^k_{r\pi_s} + \tau^k_{s\pi_r}$ is maximized, where k is the selection power. This procedure is repeated n times.

In exploring policy, once again facility r is chosen randomly uniformly. The facility s is chosen according to a stochastic rule where the probability of choosing a facility is determined by Formula

2.

$$P(s) = \frac{\tau^k_{r\pi_s} + \tau^k_{s\pi_r}}{\sum_{j \neq r}(\tau^k_{r\pi_j} + \tau^k_{j\pi_r})} \tag{2}$$

1.4 Intensification

This tool defines whether only strictly improving solutions will remain or one injects some exploration by allowing solutions, that are not the best. Initially it is activated. It is deactivated if the solutions did not a single solution changed during the current iterations, which implies that the state of stagnation was achieved. It is activated if a new best solutions was found. It corresponds to escaping a search space peak.

1.5 Pheromone update

As it was said RAS and EAS were implemented. In both of them the new pheromone trail values are defined as in Formula 3.

$$\tau(i+1) = \rho \times \tau(i) + \Delta\tau(i) \tag{3}$$

In RAS the update is done according to Formulas 4, 5, 6. The idea of the RAS is to deposit pheromones according to their rank in the sorted set of all solutions and also to the best solution.

$$\Delta\tau_{ij} = \sum_{r=1}^{w-1}(w-r) \times \Delta\tau^r_{ij} + w \times \Delta\tau^{bs}_{ij} \tag{4}$$

$$\Delta\tau^r_{ij} = \begin{cases} \frac{Q}{L^r} \text{ if arc(i,j)} \in S \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

$$\Delta\tau^{bs}_{ij} = \begin{cases} \frac{Q}{L^{bs}} \text{ if arc(i,j)} \in S^{bs} \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

, where w - is the number of depositing ants, S - current solution, S^{bs} - best solution found so far, Q - deposit factor.

In EAS the update is done according to Formulas 7, 8. The aim of this pheromone update is to deposit much larger amount pheromones to the best solution.

$$\Delta\tau_{ij} = \sum_{k=1}^{m}\Delta\tau^k_{ij} + \sigma \times \Delta\tau^{bs}_{ij} \tag{7}$$

$$\Delta\tau^{bs}_{ij} = \begin{cases} \frac{Q}{L^{bs}} \text{ if arc(i,j)} \in S^{bs} \\ 0 \text{ otherwise} \end{cases} \tag{8}$$

1.6 Diversification

Diversification is the same reset of pheromone trail values as in the initialization. The only difference is that the best solution quality found must be corrected. The pheromone trail values that is set is computed by Formula 9.

$$\tau_0 = \frac{1}{f(S^{bs})} \tag{9}$$

2 Automatic tuning

The summary of all parameters of the program is shown in Table 1.

Table 1: Parameters of the QAP solving program

Name	**Values**	**Description**
localSearch	local-search-idsia, local-search-nonet	Whether local search is used or omitted
m	1-100	Number of ants
ρ	0.0-1.0	Weakness of evaporation
roundsReinitialize	100-10000	Numbers of non-improving rounds in a row to diversify
q	0.0-1.0	Probability of using exploiting policy
k	0.2-3.0	Selection power
factorQ	0.2-5.0	Deposit factor
σ	1-100	Number of depositing for Elitist Ant System
w	1-20	Number of depositing for Rank-based Ant System

The automatic tuning was carried out by meas of the irace, developed by IRIDIA research group. It is inspired by genetic algorithms. Several initial configurations are generated and tested on the predefined problem instances. Then a new generations will be generated by inheriting the parts of the best configurations from the previous iterations. This process is repeated until few good solutions are left.

In order to make program runs more fair, we scaled the runtime of the runs. Duration of a run in seconds was assigned equal to the size of a problem. The automatic tuning was done separately for EAS/RAS. In addition we separately tune the configuration with local search and without. The resulting configuration are stored in the folder "docs/results-tuning".

3 Experimental

10 runs were performed on each of the given instances. The results are stored in "docs/results-experiments" folder in files "min.csv", "max.csv", "average.csv" and "standard-deviation" and also

shown in Tables 2 3 4 5. General sheets with computed relative solution qualities and variation coefficients are stored in the "results.xlsx" file.

Speaking of the both algorithms (EAS and RAS) with their 3 versions (without LS, with LS not tuned, with LS tuned), based on the obtained results one can say that the best performing algorithm is RAS (with LS tuned). The worst is RAS (without LS).

3.1 No local search

Both EAS and RAS without local search were tried on all the instances. EAS showed slightly better results with relative solution quality 3.75%, however it also had larger variation coefficient (meaning that some concrete instances may be solved by EAS worse than by RAS).

Wilcoxon-rank test indicated $p_{value} = 40.69\%$, which means that we cannot firmly reject the H_0 hypothesis. Means of performances of the both algorithms hypothetically can be equal.

Solution convergence process is demonstrated on Figure 1 with logarithmic scale over the evaluation axis. RAS dominates up to roughly 200 evaluation, and EAS dominates after.

3.2 With local search (not tuned)

3.3 With local search (tuned)

4 Appendix

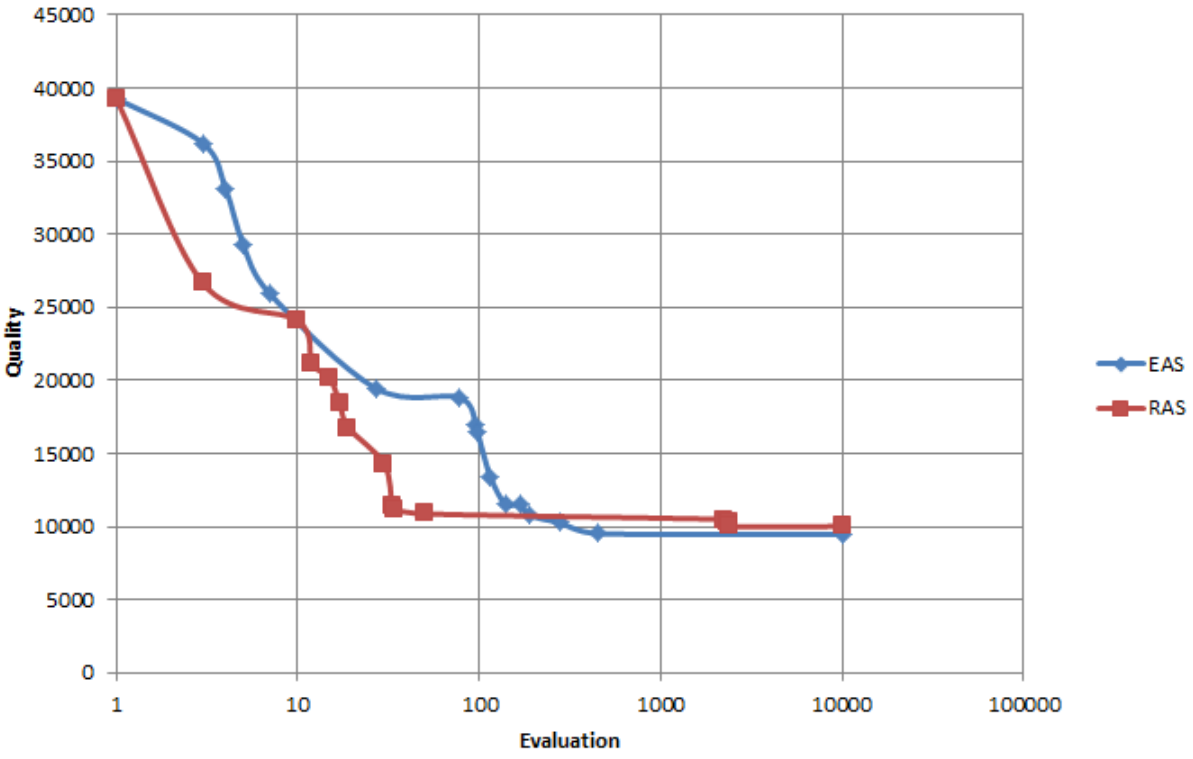

Figure 1: Performance convergence for the "chr15b.dat" instance by EAS and RAS without Local Search.

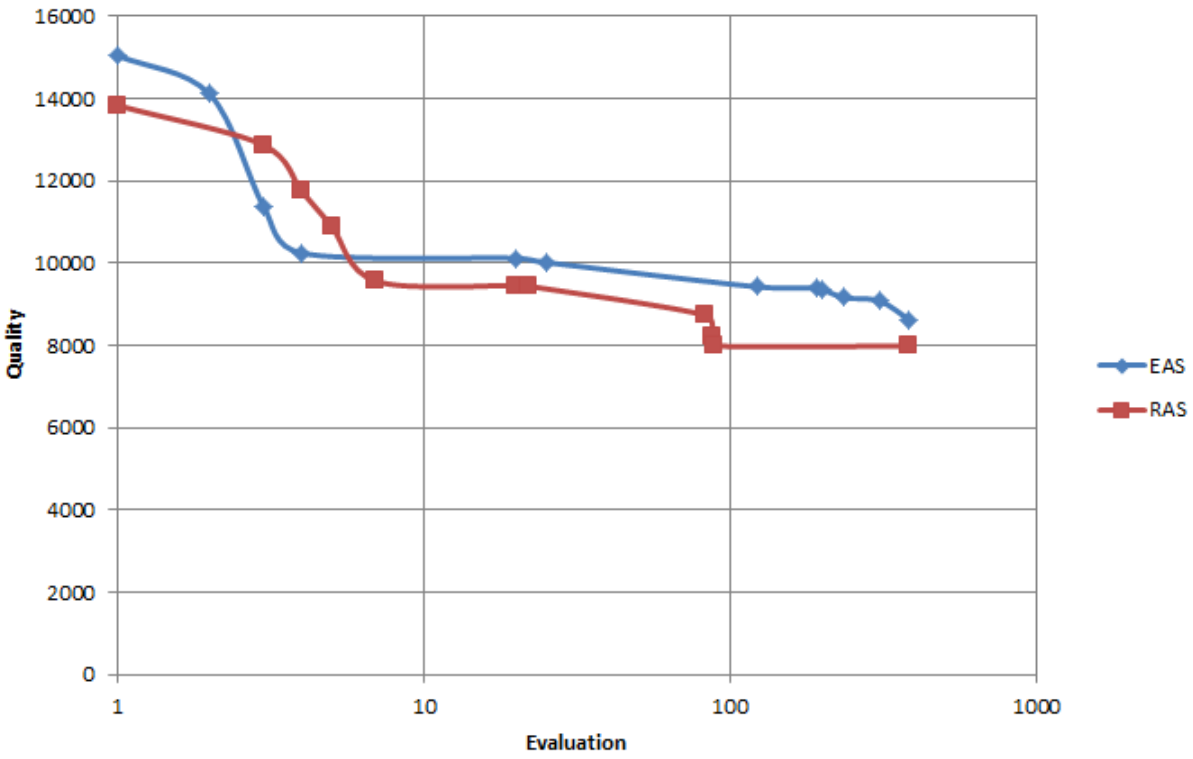

Figure 2: Performance convergence for the "chr15b.dat" instance by EAS and RAS with Local Search (not tuned).

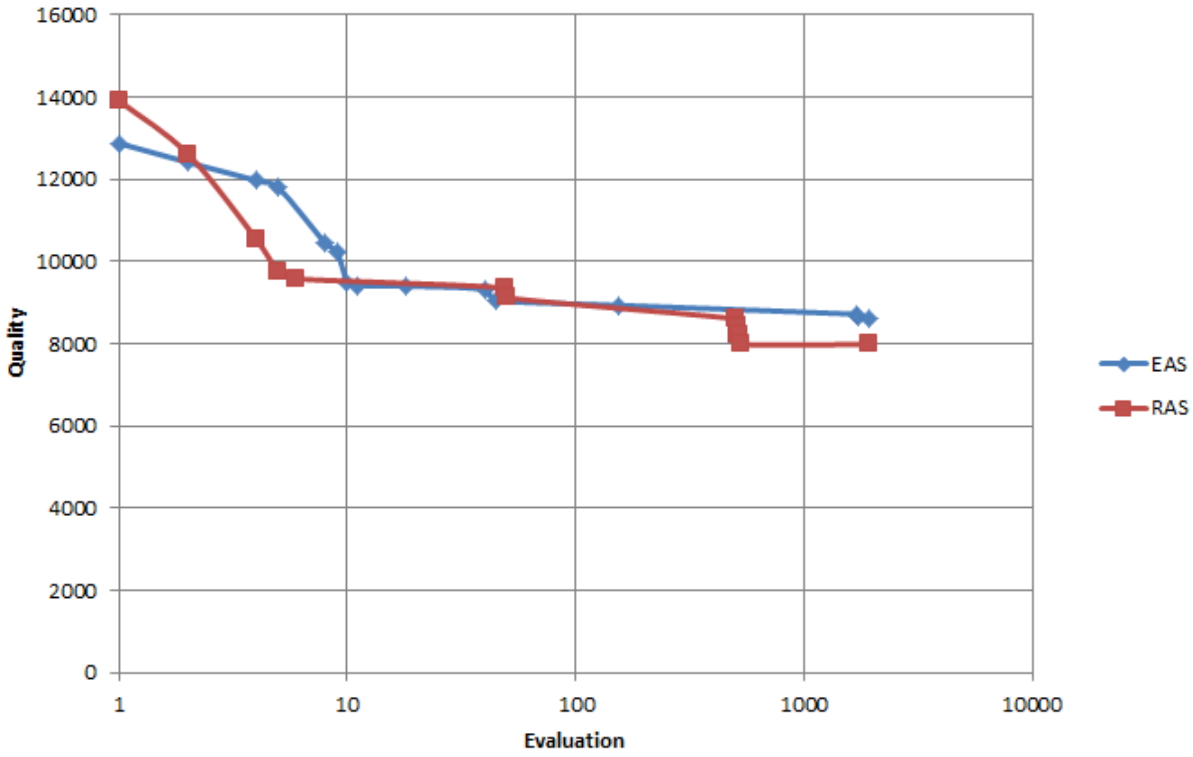

Figure 3: Performance convergence for the "chr15b.dat" instance by EAS and RAS with Local Search (tuned).

Table 2: Minimum solution qualities per each algorithm per each instance computed over 10 runs

Instance	EAS (no LS)	RAS (no LS)	EAS (LS, not tuned)	RAS (LS, not tuned)	EAS (LS, tuned)	RAS (LS, tuned)
bur26g.dat	9508105	9508105	9507884	9507884	9507884	9507884
chr15b.dat	8640	8640	7990	7990	7990	7990
esc16d.dat	16	16	16	16	16	16
had14.dat	2724	2724	2724	2724	2724	2724
had20.dat	6922	6922	6922	6922	6922	6922
nug28.dat	5252	5280	5166	5166	5170	5166
scr12.dat	31410	31410	31410	31410	31410	31410
tai17a.dat	504408	501058	493662	491812	493662	493662
tai35a.dat	2560610	2603248	2511090	2513606	2475022	2470994
wil50.dat	49520	50058	48990	48892	48896	48874

Table 3: Maximum solution qualities per each algorithm per each instance computed over 10 runs

Instance	EAS (no LS)	RAS (no LS)	EAS (LS, not tuned)	RAS (LS, not tuned)	EAS (LS, tuned)	RAS (LS, tuned)
bur26g.dat	9549292	9586656	9508853	9507884	9508853	9507884
chr15b.dat	11978	11494	9290	8626	9290	9290
esc16d.dat	16	16	16	16	16	16
had14.dat	2744	2744	2724	2724	2724	2724
had20.dat	6964	6972	6948	6922	6922	6948
nug28.dat	5442	5358	5252	5228	5272	5254
scr12.dat	32996	32958	31410	31410	31410	31410
tai17a.dat	514492	516242	499670	500790	501326	502304
tai35a.dat	2660234	2680092	2541096	2544234	2504592	2530968
wil50.dat	50354	51168	49152	49056	49028	48996

Table 4: Average solution qualities per each algorithm per each instance computed over 10 runs

Instance	EAS (no LS)	RAS (no LS)	EAS (LS, not tuned)	RAS (LS, not tuned)	EAS (LS, tuned)	RAS (LS, tuned)
bur26g.dat	9517432	9527396	9507980	9507884	9508174	9507884
chr15b.dat	10227	10093	8532	8371	8854	8512
esc16d.dat	16	16	16	16	16	16
had14.dat	2730	2734	2724	2724	2724	2724
had20.dat	6931	6942	6927	6922	6922	6929
nug28.dat	5312	5318	5201	5199	5215	5207
scr12.dat	32202	32123	31410	31410	31410	31410
tai17a.dat	510175	509739	495976	496344	498228	497675
tai35a.dat	2614268	2642542	2529051	2532861	2484877	2488667
wil50.dat	49967	50503	49067	48939	48952	48924

Table 5: Standard deviations of solution qualities per each algorithm per each instance computed over 10 runs

Instance	EAS (no LS)	RAS (no LS)	EAS (LS, not tuned)	RAS (LS, not tuned)	EAS (LS, tuned)	RAS (LS, tuned)
bur26g.dat	16737.45	28934.45	306.43	0	468.07	0
chr15b.dat	1122.67	996.02	430.1	328.43	378.98	495.72
esc16d.dat	0	0	0	0	0	0
had14.dat	9.66	10.54	0	0	0	0
had20.dat	14.34	18.4	10.96	0	0	12.59
nug28.dat	62.71	27.65	27.44	18.36	29.89	28
scr12.dat	600.72	582.08	0	0	0	0
tai17a.dat	3286.57	4681.04	2334.31	2780.92	2286.86	2986.83
tai35a.dat	30049.49	24688.77	9590.59	9326.62	10462.45	20092.04
wil50.dat	259.73	323.45	53.97	45.09	44.47	43.72