# IMPROVEMENT OF ENHANCED ANT COLONY SYSTEM FOR THE SEQUENTIAL ORDERING PROBLEM

Wojciech GRZENIA*

**Abstract.** The Sequential Ordering Problem (SOP) is a problem similar to the Asymmetric Traveling Salesman Problem (ATSP). The main difference is that the precedence constraints on the vertices mark them as forbidden in all feasible solutions. Possible real-world applications of SOP are production planning, single vehicle routing problems with pick-up and delivery constraints and transportation problems in flexible manufacturing systems. The Enhanced Ant Colony System algorithm for the Sequential Ordering Problem was studied and modified to obtain better results for the same computation time.

**Keywords:** Sequential ordering problem, SOP, Asymmetric travelling salesman, Ant colony optimization, ACS, Enhanced ant colony optimization, EACS, Enhanced ant colony optimization 2, EACS2

## 1. Introduction

In the Sequential Ordering Problem (SOP) the main goal is to find the shortest path from the given start point to the given end point which goes through all of the points exactly once. In addition there are precedence constraints which mark the connections between points as forbidden in all feasible solutions.

The SOP can model real-world problems such as production planning [2], single vehicle routing problems with pick-up and delivery constraints [6] and transportation problems in flexible manufacturing systems [1].

Gambardella et al. [5] managed to significantly increase the performance of Ant Colony System algorithm designed to solve sequential ordering problems by creating the Enhanced Ant Colony System (EACS) algorithm. This paper presents EACS algorithm modification which makes it perform even better for most of the used problem instances.

* https://uk.linkedin.com/in/wojciechgrzenia

## 2. Problem description

The SOP can be described using the graph terminology terms. A complete directed graph $G = (V, E)$ is given. Its nodes represent tasks. Each arc $(i, j)$ has a non-negative cost $t_{ij} \in R$ representing the delay between finishing the task $i$ and starting the task $j$. An additional precedence digraph $P = (V, F)$, which is defined on the same set of nodes, is given. If the arc $(i, j) \in F$ exists, it means that task $i$ must be accomplished before task $j$, otherwise the solution will not be feasible. If there are arcs $(i, j) \in F$ and $(j, k) \in F$, it implies that arc $(i, k) \in F$ also exists. Because each solution must start with task $0$ and end with task $n$, it means that $(0, i) \in F \; \forall i \in V \backslash \{0\}$ and $(i, n) \in F \; \forall i \in V \backslash \{n\}$. The target in solving SOP is to find such sequence of task execution which will have the minimum total cost while taking under consideration the precedence constraints. This means that the goal is to find Hamiltonian path in graph $G$ without violating precedence constraints defined in graph $P$.

## 3. The original EACS algorithm

The EACS algorithm is an element of the Ant Colony Optimization (ACO) algorithms family and it was introduced by Gambardella et al. [5]. The main element of this algorithms is a set of ants, which are individual small computing units. These units build solutions on their own, basing on given universe objects that represent the problem. The best solution found is returned as a result. While creating the solution each ant changes the universe by leaving information about the attractiveness of particular parts that it had contact with. This information is stored as a pheromone trail.

Ants have two ways of picking up the next part of the solution being created. They can either take the next step from the best, already found solution (if it is feasible) or the next step can be determined by using the deterministic rule. The choice is based on calculated probability. If the deterministic rule is used then each possible step is assessed and given appropriate probability of being chosen. During the assessment process two factors are taken under consideration: the heuristic function result and the pheromone amount associated with the step.

In the real world ants search for food and during that process they leave pheromone trail. Because the trail evaporates it is stronger on short paths than on the long ones. That is why ants prefer to choose routes that have strong trail. This mechanism with few changes is reflected in EACS. The stronger the trail is, the more probable it is that next ants will choose the step associated with it. In EACS the pheromone amount is changed at the end of the iteration, after all ants have built their solutions. At that point the trail is updated on the best found (during whole computation process) solution. The formula used to calculate the new value of pheromone trail is as follows:

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \rho / L_{best} \tag{1}$$

where $L_{best}$ is the total cost of the best found solution and the $\rho$ is algorithm parameter which satisfies the condition $\rho \in \langle 0,1 \rangle$. The equation consists of two parts. The $(1 - \rho) * \tau_{ij}$ is responsible for evaporating pheromone while the $\rho / L_{best}$ meets the need of increasing the pheromone amount associated with step $ij$. The pheromone amounts are also updated during

solution creation. In that case they are decreased to make it less probable for the following ants to use the same steps in theirs solutions.

When an ant generates a full path with a total cost less than or equal to $1{,}2 * L_{best}$, the local search process is applied to that solution. This process intention is to create even better solution by introducing slight changes to the given path. This is achieved by changing three arcs and verifying whether generated solution is getting better. This process is iterative. An ant holds the information about all nodes in the solution that still needs to be used as a starting point for change analysis in a *don't-push-stack* first introduced by Gambardella et al. [5]. The solutions generated during EACS computing are generally very similar to the best found solution. Often large sequences of steps are reflected in following attempts to find the new best path. To avoid situation when the same changes are verified multiple times during local search, the EACS algorithm performs the check only for the parts of the solution that "are out of sequence with respect to the best solution retrieved so far". Unfortunately this statement is not precise enough to deduce what exactly should be in the stack, so during the research work assumption was made, just like in Ezzat A. work [3], that if element $i$ in the tour being improved and the best found tour are different, then $i$ is considered out of sequence.

This description of EACS algorithm presents only its most important aspects in relation to the modifications presented in this article. Gambardella et al. present more details in their work [4], [5].

## 4. The EACS2 algorithm

During the research multiple modifications of the original EACS algorithm were tested. The most promising results were achieved by combining five of them:

- initial value of pheromone on each arc depends on its cost,
- decreasing pheromone trail during local search process on each new arc that is checked,
- swapping the best found solution in case of finding the solution with exactly the same cost,
- stack initialization during local search process based on arcs that are not present in the best found solution,
- no pheromone reduction for the best found solution.

### 4.1. Initial value of pheromone on each arc depends on its cost

This modification presents a new formula to calculate the initial value of pheromone on each arc:

$$\tau_{ij} = \tau_{init} * (t_{ij}/t_{favg}) \tag{2}$$

The $\tau_{init}$ is calculated from $\tau_{init} = (FS * n)^{-1}$, where $FS$ is the cost of the best solution generated by running one iteration of the algorithm without local search. The $t_{favg}$ is an average cost of arcs used to create the $FS$ solution.

This modification leads to lowering the impact of heuristic function in the early stages of computing by increasing the initial attractiveness of costly arcs over the cheap ones. The algorithm start the search from different areas than the traditional EACS would. The change of pheromone initialization stops algorithm from always rejecting the solutions which start from costly arcs and end with cheap ones. These solutions can also be very attractive, but ants in the EACS algorithm are very unlikely to pick costly arcs in the initial phase of creating the solution because of very low heuristic function assessment result.

### 4.2. Decreasing pheromone trail during local search process on each new arc that is checked

When an ant uses an arc to create a solution it consumes a bit of pheromone on that arc. This reduces its attractiveness and makes it less probable that the same arc will be pick up by other ants. In the original EACS ant does not consume pheromone on all used arcs. The consumption occurs only for arcs that are used when creating the initial solution, but arcs checked during local search are not taken into account.

Presented modification changes the behaviour of ants, so that they consume the pheromone also during the local optimization phase. This creates even more diversity in produced solutions. The consumption occurs only for the arcs that are not present in the improved path.

### 4.3. Swap the best found solution in case of finding the solution with exactly the same cost

In the original EACS a solution is marked as best when its cost is lower than the cost of current best one. Presented modification changes this condition, so that a solution is marked as best if its cost is lower or equal to the previous best one. The goal is to extend the search area on which algorithm operates. The best found solution is used a lot to create next generation paths. The gain of swapping that solution to the one with exactly the same potential is that the search area moves a bit and generating paths that were already checked is less probable.

### 4.4. Stack initialization during local optimization process based on arcs that are not present in the best found solution[1]

In the EACS algorithm the local search process starts with initialization of the *don't-push-stack*. It is filled with elements that are considered out of sequence in relation to the best found solution. Element $i$ of the improved solution is considered out of sequence when it is different than element $i$ of the best found solution. With this approach however it is possible

[1] As it was mentioned before in section 3 the description of stack initialization during local search in [5] was not precise which means that presented modification might actually be the right way of implementing the EACS algorithm.

that the optimization process will be performed multiple times for the same sequence of elements. This is demonstrated by the example. Two solutions are given: one that is being improved $A$ and the best found $B$. Figure 1 presents the initialization process of stack $H$ in the EACS algorithm.

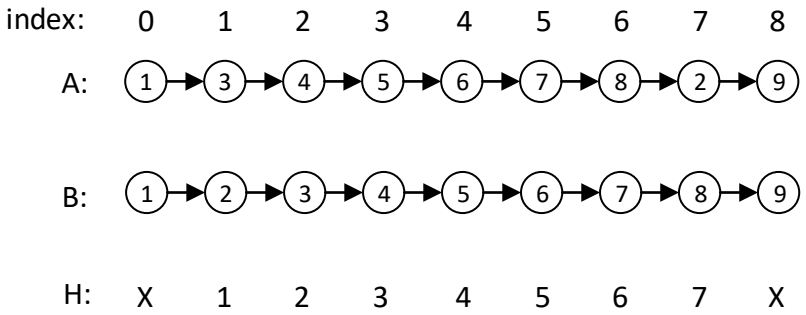


**Figure 1. Initialization process of *don't push stack* during local search in the EACS algorithm**

In this situation the stack contains almost all indexes, however the route from node 3 up to node 8 is exactly the same in both solutions. All changes made between indexes 1 and 6 were already processed by local optimization while creating the solution $B$ and it didn't bring any improvement (if it did then the change would have been applied and $B$ would not contain this sequence).

The proposed modification is to change the stack initialization so that the stack contains only the nodes that are parts of arcs not present in the best found solution. Figure 2 contains an example of modified stack initialization.

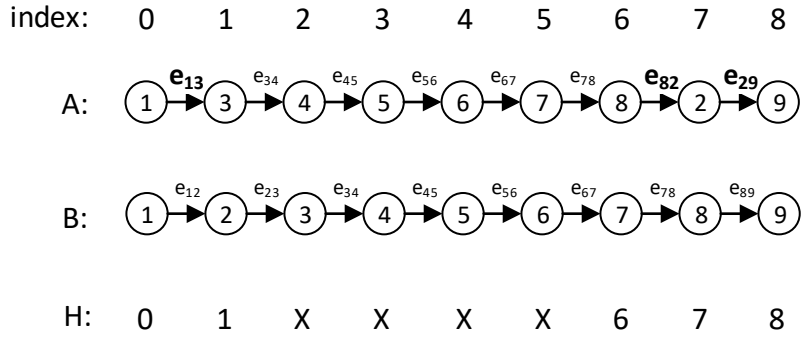


**Figure 2. Initialization process of *don't push stack* during local search in the EACS2 algorithm**

In the Figure 2 arcs were marked with letter $e$. Each arc in the solution $A$ which is not present in $B$ was made bold.

### 4.5. No pheromone reduction for the best found solution

In the EACS the process of global pheromone update is performed only on the best found solution and because of that it always implies both phases:

- pheromone addition,
- pheromone evaporation.

The pheromone addition phase is dominant which means that more pheromone is added to the arc than evaporated. This leads to a conclusion that the evaporation part is redundant. The comparison tests of the two versions of EACS – with and without evaporation during global pheromone update – showed no significant differences in algorithms performance. As removing the pheromone evaporation makes the algorithm a bit simpler, the decision was made to use this modification.

## 5. Experimental results

### 5.1. Test program

The test program was implemented using C# language and .NET framework. It contains my implementation of EACS algorithm and multiple modifications which can be turned on or off. In case when arc cost $t_{ij}$ equals zero, the heuristic function, which normally is calculated from equation: $\eta_{ij} = 1/t_{ij}$, will return value of 2.

### 5.2. Test data

The benchmark was performed using the SOPLIB2006 library[2], commonly used in other publications related to SOP. This library contains 48 SOP instances, each in separate file with *.sop* extension. Each file name uses R.*n.r.p* template, where

- *n* stands for number of nodes;
- *r* stands for maximum arc cost;
- *p* stands for approximate percentage of precedence constraints, which means that number of precedence constraints for the problem will be approximately

$$(p/100) * (n * (n-1)/2)$$

### 5.3. Test environment

Implemented test program was compiled in Release mode for 64-bit processors. All benchmarks were performed using computer with Windows 8.1 operating system, Intel i7-2600K processor with clock set to 3.74 GHz. Implemented program allows

[2] http://www.idsia.ch/~roberto/SOPLIB06.zip

multithreading. The i7-2600K processor has 8 logical cores. During benchmarks maximum of 5 were used at once.

## 5.4. Benchmark results

The algorithm parameters used in benchmark are those already used in Gambardella and Dorigo paper [4]. The results presented in Table 1 were obtained after 10 runs with maximum computation time set to 600 seconds.

**Table 1. Benchmark results for EACS and EACS2 algorithms**

| | *Average cost* | | |
|---|---|---|---|
| *File* | EACS | EACS2 | Improvement |
| *R.200.100.1.sop* | 74.9 | 69.6 | 7.08% |
| *R.200.100.15.sop* | 1925.5 | 1937.8 | -0.64% |
| *R.200.100.30.sop* | 4236.2 | 4235.7 | 0.01% |
| *R.200.100.60.sop* | 71749 | 71749 | 0.00% |
| *R.200.1000.1.sop* | 1471.5 | 1465 | 0.44% |
| *R.200.1000.15.sop* | 22362.9 | 22098.1 | 1.18% |
| *R.200.1000.30.sop* | 41353.3 | 41363.8 | -0.03% |
| *R.200.1000.60.sop* | 71570.5 | 71556 | 0.02% |
| *R.300.100.1.sop* | 46.2 | 35.1 | 24.03% |
| *R.300.100.15.sop* | 3528.1 | 3539.3 | -0.32% |
| *R.300.100.30.sop* | 6204.3 | 6183.3 | 0.34% |
| *R.300.100.60.sop* | 9726 | 9726 | 0.00% |
| *R.300.1000.1.sop* | 1462.9 | 1449.5 | 0.92% |
| *R.300.1000.15.sop* | 33301.4 | 33521.7 | -0.66% |
| *R.300.1000.30.sop* | 55545.1 | 55688.6 | -0.26% |
| *R.300.1000.60.sop* | 109783.9 | 109748.2 | 0.03% |
| *R.400.100.1.sop* | 39.9 | 28.4 | 28.82% |
| *R.400.100.15.sop* | 4761.9 | 4703.1 | 1.23% |
| *R.400.100.30.sop* | 8511 | 8395.5 | 1.36% |
| *R.400.100.60.sop* | 15273.6 | 15281.6 | -0.05% |
| *R.400.1000.1.sop* | 1575.4 | 1510.8 | 4.10% |
| *R.400.1000.15.sop* | 46890.7 | 46490.6 | 0.85% |
| *R.400.1000.30.sop* | 87839.1 | 87323.3 | 0.59% |
| *R.400.1000.60.sop* | 141531.9 | 141338.4 | 0.14% |
| *R.500.100.1.sop* | 34.9 | 20.4 | 41.55% |

| | | | |
|---|---|---|---|
| *R.500.100.15.sop* | 6701 | 6625.4 | 1.13% |
| *R.500.100.30.sop* | 10336.8 | 10206.5 | 1.26% |
| *R.500.100.60.sop* | 18407.9 | 18382.5 | 0.14% |
| *R.500.1000.1.sop* | 1683.1 | 1540.5 | 8.47% |
| *R.500.1000.15.sop* | 62426.6 | 63036.1 | -0.98% |
| *R.500.1000.30.sop* | 104370.2 | 103732.2 | 0.61% |
| *R.500.1000.60.sop* | 179575.3 | 178868 | 0.39% |
| *R.600.100.1.sop* | 29.2 | 17.1 | 41.44% |
| *R.600.100.15.sop* | 7650.8 | 7498.4 | 1.99% |
| *R.600.100.30.sop* | 13411.5 | 13095.7 | 2.35% |
| *R.600.100.60.sop* | 23996.6 | 23606.4 | 1.63% |
| *R.600.1000.1.sop* | 1799.9 | 1639.3 | 8.92% |
| *R.600.1000.15.sop* | 74328.1 | 73133.5 | 1.61% |
| *R.600.1000.30.sop* | 137129.7 | 134761.5 | 1.73% |
| *R.600.1000.60.sop* | 224034.2 | 220308.7 | 1.66% |
| *R.700.100.1.sop* | 26.6 | 17.1 | 35.71% |
| *R.700.100.15.sop* | 9604.7 | 9404.2 | 2.09% |
| *R.700.100.30.sop* | 16282.2 | 15659.6 | 3.82% |
| *R.700.100.60.sop* | 25309.8 | 24736 | 2.27% |
| *R.700.1000.1.sop* | 1750.2 | 1605.1 | 8.29% |
| *R.700.1000.15.sop* | 92592 | 89502.4 | 3.34% |
| *R.700.1000.30.sop* | 148859.8 | 147155.1 | 1.15% |
| *R.700.1000.60.sop* | 257040.2 | 253755.1 | 1.28% |
| | | **Average:** | **5.02%** |

Table 2 shows the average improvement for problems grouped by *p* parameter.

**Table 2. Average improvement for problems grouped by *p* parameter**

| *p* | *Average improvement* |
|---|---|
| *1* | 17.48% |
| *15* | 0.90% |
| *30* | 1.08% |
| *60* | 0.63% |

The results from Table 1 show that EACS2 performed better than EACS for 39 out of 48 SOP instances. For two problems results obtained were exactly the same. The average improvement of applying modifications was 5.02%. The results presented in Table 2 indicate that the greatest improvement was achieved for problems with small ratio of precedence constraints – 17.48%. For the rest of the problems the quality of results was 0.87% better.

## 6. Conclusions

A modified version of Enhanced Ant Colony System algorithm (EACS2) for the Sequential Ordering Problem was presented. The EACS2 algorithm was obtained by applying five different modifications. Experimental results have shown that EACS2 performs better than EACS in most cases, giving an average improvement of 5.02%. The improvement varies a lot between groups of problems with similar ratio of precedence constraints. It was also observed during studies of modifications not presented in this paper. This leads to a conclusion that further investigation of dependencies between modifications and ratio of precedence constraints could lead to creating efficient algorithm which behaviour would be highly dependent on the number of precedence constraints.

## References


[1] Ascheuer N., *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*, PhD thesis, Technische Universität, 1995

[2] Escudero L. F., An inexact algorithm for the sequential ordering problem, *European Journal of Operational Research*, **37**, 2, 1988, 236-249.

[3] Ezzat A., *Ant Colony Optimization Approaches for the Sequential Ordering Problem*, Master of Science Thesis, The American University in Cairo, Department of Computer Science and Engineering, 2013. http://dar.aucegypt.edu/bitstream/handle/10526/3629/Ant%20Colony%20Optimization%20Approaches%20for%20the%20Sequential%20Ordering%20Problem.pdf?sequence=1

[4] Gambardella L. M., Dorigo M., An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, *INFORMS Journal on Computing*, **12**, 3, 2000, 237-255.
[5] Gambardella L. M., Montemanni R., Weyland D., An Enhanced Ant Colony System for the Sequential Ordering Problem, *Operations Research Proceedings 2011*, 2011, 355-360.
[6] Savelsbergh M. W. P., An efficient implementation of local search algorithms for constrained routing problems, *European Journal of Operational Research*, **47**, 1, 1990, 75-85.