**Multi-Objective Public Transportation Traffic Network Optimization**

Baron Yotam & Malka Amit

**Abstract**

Planning a useful public transportation traffic network is a difficult and an important task for the well-being of citizens. It is a difficult task since several objectives need to be optimized simultaneously, there are constraints and the number of possible solutions to the task is enormous. In this paper, we implemented and tested three different algorithms. The algorithms were based on a Simulated Annealing (SA) algorithm, a Genetic algorithm (GA) and an Artificial Bee Colony (ABC) algorithm. We had in mind four objectives to optimize i.e., minimal distance and travel time driven by the buses and maximal connectivity of the whole network and bus visits to stations that passengers arrive at frequently. We used a dominance method to obtain a pareto front of solutions from every algorithm, where we later showed and evaluated them against one another. We tested the algorithms on five networks that differentiated from one another in the number of nodes (stations), the number of buses available and the minimum and maximum number of stations that each bus had to visit in its planned path. Results showed that the Genetic algorithm was the most efficient in terms of run time, however it achieved the worst results in terms of the solution's evaluated measures with comparison to the ABC and SA algorithms, where the latter was superior in most cases and the most stable in terms of run time.

***Key words: Simulated Annealing, Genetic algorithms, multi-objective optimization, pareto front***

# Introduction

The Public transportation network system is composed of four categories: facility network, route network, organization network and demand network. The intertwined networks form the essential base for the human interaction in social and economic activities and even the urban architecture. In the context of transportation network, the traffic nodes constitute a facility network, the traffic lines make up a route network, and the combination of nodes and lines forms organization network, including road network, public transportation network, urban external transportation network, and passenger and cargo transportation hubs. Public transportation network is an important issue for human activities. Public transportation network design problem (PTNDP) is so complicated and practically important that it has been studied since the last five decades. As a result, more researchers have published increasingly growing works over time. Some reviews among these researches have been conducted by Migdalas [[1](#_References)], Yang and H. Bell [[2](#_References)], Kepaptsoglou and Karlaftis [[3](#_References)], Farahani and Miandoabchi et al. [[4](#_References)], and by Xu and Chen et al. [[5](#_References)].

In this paper we represent the four categories in PTNDP by multi-objective optimization problem. The multi-objective optimization problem (also known as multi-objective programming problem) is a branch of mathematics used in multiple criteria decision-making, which deals with optimization problems involving two or more objective function to be optimized simultaneously [[6](#_References)]. The objective is to find the bus paths that minimize distance and travel time of the buses, minimize the frequency of passengers arriving at the station and maximize connectivity of the network. Using weighted edges and nodes helps us to explore the influence of those functions. Nodes are stations and edges represent a transition that a certain bus includes in its path from node to node. We apply two weights in each edge, the first represents distance and the second travel time. In addition, each node gets one weight representing the frequency of passengers arriving at that station. At last, we measured a global network weight representing the connectivity of the network, if passengers can reach every station from every station. The multi-objective function solution is to find the bus paths that optimize these four weights simultaneously. The solution had to uphold two constraints - every station in the network is visited at least once in the entire transportation scheme and the same bus can't visit the same station twice (no circles).

In all of our algorithms' solutions we used the method of Pareto front which deals with multi-objective functions. The concept of Pareto front or set of optimal solutions in the space of objective functions in multi-objective optimization problems (MOOPs) stands for a set of solutions that are non-dominated to each other but are superior to the rest of solutions in the search space. It means that it is not possible to find a single solution that is superior to all other solutions with respect to all objectives, so that changing the vector of design variables in such a Pareto front consisting of these non-dominated solutions could not improve all objectives simultaneously. Therefore, such a change will worsen at least one objective. Thus, each solution of the Pareto set includes at least one objective inferior to another solution in that Pareto set, although both are superior to others in the rest of search space [[7](#_References)].

Due to the computational complexity of the multi-objective public transportation problem, the most related research concentrates on heuristic-based algorithms, especially the meta-heuristic approaches aiming to find near-optimal solutions. The Simulated annealing (SA) algorithm was one of those meta-heuristics. SA is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a meta-heuristic to approximate global optimization in a large search space for an optimization problem. In addition, we used a Genetic algorithm (GA), another meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection [[8](#_References)]. The last meta-heuristic we used is the Artificial Bee Colony (ABC) algorithm that was defined by Karaboga and Basturk [[9](#_References)], motivated by the intelligent behavior of honeybees and has been applied to solve many problems and obtained satisfying results. In this paper, all algorithms are extended with Pareto based multi-objective to solve the PTNDP. Section ‎2 mathematically formulates the public transportation traffic network problem. An expansion on the algorithms and on the experiments we conducted to evaluate them will be given in Section [3](#_Method). Section [4](#_Results) summarizes the results and Section [5](#_Conclusion_+_Future) summarizes the contribution of this paper along with some future research directions.

# Public transportation traffic network problem

## 2.1 Problem definition

For a given traffic network which is modeled as a undirected graph G = (N, E), where N is the set of nodes and E is the set of links, the Public transportation traffic network Problem in this paper is to find a bus paths that meets different optimization criteria and satisfies the specified constraints. The attribute distance, travel time and frequency of passengers arriving at the station and network connectivity related criteria are considered simultaneously in this paper, and the problem is formulated as follows:

Where:

(1)

(2)

(3)

(4)

Subject to the constraints:

The,, and are the objective functions related to the distance, travel time, frequency, and network connectivity attributes respectively. The p denotes a bus paths solution graph, having only edges that were found in the bus paths solution. The , and denotes the weights of distance, travel time, frequency, and network connectivity attributes respectively, these weights are given by the user or set uniformly in case no such information exists. The and denote the distance and travel time of the link (i, j) ∈ E respectively. The denotes the passengers' arrival frequency at station *i*. The is a binary variable, that takes the value of 1 if the link (i, j) ∈ E is used in bus paths; otherwise, it takes the value of 0. In addition, the is also a binary variable, that takes the value of 1 if is used in bus paths; takes the value 0 otherwise. is a connectivity measure that counts the number of two-stations that cannot be reached from one another in the given bus paths divided by the number of all two-stations combinations in the entire network. The first constraint is implemented to make sure every station in the network is visited at least once by any bus. The second constraint makes sure that the same bus can't visit the same station twice (no circles). The inputs from the user to our algorithms are the number of buses to find optimal paths for, the number of minimal and maximal stations each bus needs to go through, the number of stations in the network, the distance and travel time between every two stations and the frequency of arriving passengers to every station.

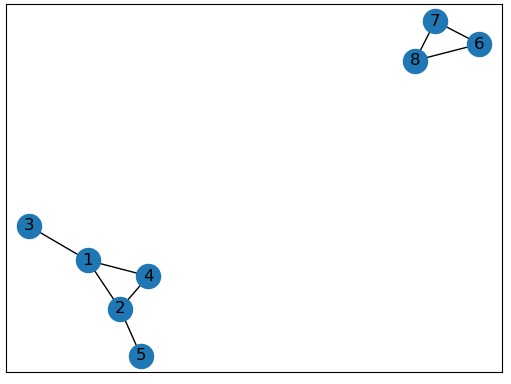
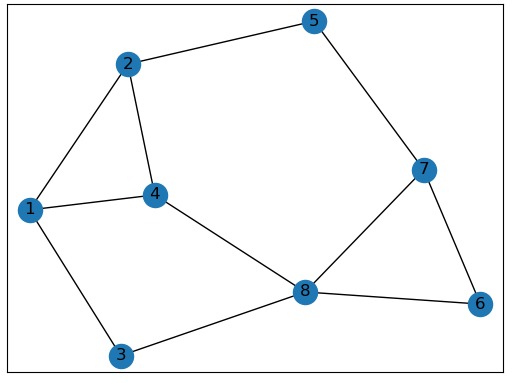
# 

# Method

We implemented and tested three algorithms; a Simulated Annealing algorithm, a Genetic algorithm, and an Artificial Bee Colony algorithm, we will elaborate on each algorithm in the next section. In order to test our algorithms, we created five experiments that are based on synthetic data.

Every experiment is based on a traffic network. The network's nodes represent stations, and each node holds a weight that represents how frequently passengers arrive at that station. The networks edges represent transitions between two stations, and each edge holds two weights; the first weight represents the distance between the two stations and the second weight is the travel time between the stations. Each one of these three weights were synthetically sampled with a uniform distribution between 0 and 1.

Our objective was to find the bus paths for a given number of buses that minimize the total distance and travel time weights and minimizes the frequency weight multiplied by minus 1. Another weight that we tried to minimize was the connectivity of the network established by the bus paths, we calculated the number of pairs of stations that cannot be reached from one another and divided it by the total number of pairs of stations to get a weight between 0 and 1. Figure 1a shows a network that is fully connected and figure 1b shows a network that is partially connected. Every bus path length had to stay between a given minimal and maximal number of stations. Figure 2 shows a summary of the objective function weights.

**Figure 1a. (left) a fully connected network. Figure 1b. (right) a partially connected network**

|  |  |  |  |
| --- | --- | --- | --- |
| **Objective criterion** | **Meaning** | **Range** | **Min / Max** |
| Distance | The total distance weight covered by all buses in their paths (weights are on the edges) | [0, 1] | Min |
| Travel time | The total travel time weight covered by all buses in their paths (weights are on the edges) | [0, 1] | Min |
| Arrival frequency | A sum of the visited nodes' weight – every node that is visited by a bus in its path is accounted for (a node can be summed more than once) | [0, 1] | In general – max. But we wanted to minimize all weights, so we minimized the multiplication of this weight by minus 1 |
| Connectivity | The number of pair of nodes that cannot be reached from one another by the traffic network built from the buses' paths divided by the total number of two nodes combinations | [0, 1] | Min |

**Figure 2. Objective function's weights summary**

Figure 3 shows the inputs we used for each experiment. We ran the experiments in two manners and evaluated them differently. First, we created 10 permutations for each experiment, meaning that we created 10 networks for each network size – each had different distance, travel time and frequency weights sampled by changing seeds. The importance that we used for each of the four objective function criteria was set uniformly – 0.25 for each criterion. This way allowed us to get one score for the objective function – the average of the four weights. The results were averaged for each algorithm in each experiment over the 10 permutations. The evaluation measures that we used in this part were:

- Min, max and average runtime of all permutations.

- Average best score of all permutations.

- Average number of dominating solutions and dominated solutions between the algorithms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Network** | **Number of nodes** | **Number of buses** | **Min number of stations** | **Max number of stations** |
| Very small | 10 | 3 | 4 | 6 |
| Small | 20 | 5 | 4 | 7 |
| Medium | 50 | 10 | 5 | 10 |
| Big | 100 | 15 | 7 | 13 |
| Large | 200 | 20 | 10 | 15 |

**Figure 3. Experiments' inputs**

The second manner in which we ran the experiments was to test the algorithms in changing importance of the four objective function criteria. Each criterion was assigned with an importance from the following group [0.1, 0.3, 0.5, 0.7] with the exception that the sum of all criteria had to be 1, giving us 21 importance combinations – all combinations can be seen in Appendix A. We ran each importance combination once on every network size with each of the algorithms. The average score of the 21 combinations runs on each network size was the measure we used to evaluate the algorithms. In addition, we used this method to evaluate which parameters value of every algorithm give the best results.

Inputs to all algorithms:

* A traffic network with the wanted number of nodes and all weights i.e., distance, travel time and station arrival frequency.
* Objective function weights – uniform weights are the default.
* Number of buses to find paths for.
* Minimum and maximum number of stations in each bus path.

## 3.1 Simulated Annealing algorithm

Input to the algorithm:

* TI – the initial temperature for the simulated annealing process.
* TL – temperature length, which is the number of iterations to do in each temperature.
* Decrease rate – after every TL iterations the current temperature is multiplied by this decrease rate (making it less likely to accept inferior solutions in the future).
* Number of rejected solutions to stop the process.
* Temperature threshold to stop the process.

Stopping criteria:

* If the temperature drops below the temperature threshold.
* If the number of solutions that are rejected crosses the given threshold.

Output of the algorithm:

* In case we use the archive option, the algorithm returns all solutions that it reached during the process and are not dominated in every objective function criterion by another reached solution.
* In case we use the option of known objective function weights given by a user, the algorithm returns the best solution achieved with the given weights.
* The solution is the bus path of all buses in our network.
* Total run time.

The Simulated Annealing algorithm includes four steps:

1. Find a feasible initial single solution.

2. Try to improve the current solution by making four changes to it sequentially:

* Delete a station from a randomly chosen bus path.
* Add a station that doesn't already exist to a randomly chosen bus path.
* Swap an existing random station in a randomly chosen bus path with a station that is not already in the chosen path.
* Swap the positions of two stations in the same randomly chosen path.

\* Another change was tested – swapping between two random stations in different paths, but we observed that results were always worse when this change was used and thus was excluded.

3. Update the archive of seen solutions by the dominance rule.

4. Decide whether to continue with the changed solution to the next iteration or stay with the old one (the simulated annealing part of the algorithm).

\* Steps 2, 3 and 4 are executed iteratively until stopping criteria is met.

Expending on each step of the algorithm:

Step 1: We create a single initial solution to be used by the algorithm. The initial solution withholds both of our constraints i.e., all stations are visited at least once, and the same bus doesn't visit the same station twice in its path. In order to build the bus paths, we first choose a starting station for each bus. The starting station for each bus is chosen probabilistically where each station has a probability that is equal to its relative arrival frequency weight compared to the other stations. We make sure that each bus starts from a different station. After that, we construct the path for each bus individually, the next station is chosen again by a probabilistic procedure that calculates the score of going from the starting station to each other possible station – the score is the weighted sum of distance, travel time and arrival frequency and the possible stations are only those that the bus hasn’t visited yet in its path. The rest of the stations are chosen in the same manner until the path has the average (rounded down) of the minimum and maximum number of stations possible.

Having all initial bus paths at hand, we make sure that all stations are visited at least once. We calculate the frequency of the stations visited and replace stations that weren't visited at all with stations that were least visited but were visited at least twice – this is because these stations won't cause other stations to become unvisited, and they are probably the worse stations to visit since they were least visited.

Step 2: Having the initial solution at hand, we iteratively try to improve it by making four changes to it. The first change is to choose a random bus path and to delete from it the station that will give the best total score after it is deleted. The second change is adding a random station to a randomly chosen bus path in a position that is chosen again by the best score of the network after the station's addition. The third change is to choose a random bus path, choose a random station in it and replace it with a random station that is not already in it. The last change is to swap the positions between two random stations in a randomly chosen bus path.

Step 3: After receiving the changed solution, we compute its score and check whether to insert it into the archive. A solution will be added into the archive in case that it is not dominated in all objective function criteria by any other solution already in the archive.

Step 4: In case the changed solution is better than the current solution, the current solution becomes the changed solution, and another iteration begins (back to step 2). In case the changed solution is not better than the current solution, the current solution becomes the changed solution with a probability that decreases as the iterations grow. The probability is computed based on the difference in scores between the current and the changed solution (delta) and according to a probability function that incorporates the computed delta and the current temperature. This is mathematically formulated as follows:

At the first iteration, the current temperature *T* is set to be TI and after each TL iterations with the current temperature *T*, the temperature is decreased by the decrease rate.

The process stops if the temperature drops below the temperature threshold or if the number of unaccepted changed solutions exceeds a given threshold.

## 3.2 Artificial Bee Colony algorithm

The ABC algorithm attempts to mimic the natural behavior of honeybees in their search after resourceful food sources. This procedure includes employed bees that randomly find the initial food sources to evaluate and onlooker bees that each choose one of the food sources found by the employed bees and try to find neighboring food sources that are better than the current one. In case the onlooker bee finds a better neighbor, the employed bee makes the neighbor food source their current food source to gather nectar from. In case the onlooker bee cannot find a better neighbor after several attempts, the employed bee abandons its current food source and randomly finds a completely new food source. This process repeats itself.

Input to the algorithm and stopping criteria:

* Number of employed bees.
* Number of onlooker bees.
* Limit – the number of iterations for attempting to improve a solution before it is forgotten and a completely new solution is initialized.
* Number of iterations – the process stops if this number of iterations is reached.
* Best score improvement limit – the process stops if no improvement was found to the best solution after this number of iterations.

Output of the algorithm:

* In case we use the archive option, the algorithm returns all solutions that it reached during the process and are not dominated in every objective function criterion by another reached solution.
* In case we use the option of known objective function weights given by a user, the algorithm returns the best solution achieved with the given weights.
* The solution is the bus path of all buses in our network.
* Total run time.

The ABC algorithm includes seven steps:

1. Find feasible initial solutions (one initial solution for each employed bee).

2. Normalize the scores of all employed bees.

3. Assign each onlooker bee to an employed bee solution (probabilistically according to how good the solution is).

4. Onlooker bees attempt to improve the solution they were assigned to.

5. Employed bees attempt to improve their solutions.

6. Re-initialization of each employed bee solution is done if the employed bee's solution is not improved after 'limit' iterations.

7. Update the archive of seen solutions using the dominance rule.

Steps 2 through 7 are done iteratively until one of the stopping criteria is met.

Expending on each step of the algorithm:

Step 1: This step is done in the exact same way it is done in the first step of the SA algorithm. The only difference is that we repeat this step for every employed bee, meaning that every employed bee has its own initial solution, and the number of initial solutions is equal to the number of employed bees. In addition, the number of stations in all paths at each initial solution is set to be the maximum number of stations possible.

Step 2: At this step we calculate the score of every employed bee's solution, and we normalize every score with respect to all other scores. This is done by dividing each score by the sum of all scores, thus creating higher probabilities for better scores. In addition, the normalized scores sum to one.

Step 3: The number of onlooker bees is equal to the number of employed bees. At this step each onlooker bee is assigned to one of the employed bee's solutions. The onlooker bees choose a solution by the probabilities calculated in step 2; better scoring solutions will be chosen by the onlooker bees more frequently.

Step 4: After each onlooker bee is assigned to an employed bee's solution, it tries to improve it by changing one randomly chosen path in it. The new path is constructed in the same way the initial paths are constructed (step 1). The only exception is that we construct paths in different lengths – between the minimum and maximum number of stations every bus needs to visit, eventually we insert the path with the length that achieves the best score.

Step 5: After the onlooker bees try to improve the solutions, each employed bee tries to improve its own solution. The employed bees do that by changing one randomly chosen path in their solution – they choose a random station position in the path, keep the stations before the chosen position, change the path starting with the chosen position. The same path length procedure is used here as in step 4, meaning we construct paths with length between the minimum and maximum number of stations and choose the one that gives the best score.

Step 6: After the onlooker bees and the employed bees try to improve their solution, we check whether they succeeded. We keep a counter for each employed bee's solution, and we increase it by one in case the bees were not successful in improving it. If the counter of a specific solution exceeds the 'limit' than we set the counter to zero and we re-initialize this specific solution, meaning that the current solution is completely abandoned and a new one is constructed in the same way done in step 1.

Step 7: After receiving the changed solution, we compute its score and check whether to insert it into the archive. A solution will be added into the archive in case that it is not dominated in all objective function criteria by any other solution already in the archive.

Steps 2 through 7 are done iteratively.

The process stops if the number of iterations exceeds the total number of iterations threshold or if the number of iterations with no improvement to the best solution exceeds the threshold.

## 3.3 Genetic algorithm:

Input to the algorithm and stopping criteria:

* Population – Initial solution space, number of solutions to keep in every stage.
* FITNESS-FN – score function to be calculated at every iteration.
* Number of iterations – stopping criteria.

Output of the algorithm:

* In case we use the archive option, the algorithm returns all solutions that it reached during the process and are not dominated in every objective function criterion by another reached solution.
* In case we use the option of known objective function weights given by a user, the algorithm returns the best solution achieved with the given weights.
* The solution is the bus path of all buses in our network.
* Total run time.

The Genetic algorithm steps:

1. Find feasible initial solutions (Number of initial solutions is set to be the population size).
2. Calculate the normalized score of each solution in population to represent the weights.
3. Select probabilistically by the computed weights two different solutions from population.
4. Reproduce a child from this two solutions.
5. With a small probability – insert a mutation to the child solution.
6. Add child into new population.
7. Update archive using the dominance rule.
8. The population size is the number of repetitions done of steps 2-7, after the last repetition, replace population with new population.

Steps 2 through 8 are done iteratively until the stopping criteria is met.

Expending on each step of the algorithm:

Step 1: This step is done in the exact same way it is done in the first step of the ABC algorithm. The only difference is that the number of initial solutions is set to be the population size.

Step 2: At this step we calculate the score of each solution in population, and we normalize every score with respect to all other scores. This is done by dividing each score by the sum of all scores, thus creating higher probabilities for better scores. In addition, the normalized scores sum to one.

Step 3: At this step we select two different solutions from population to represent two parents. These parents solutions are chosen probabilistically with respect to their normalized scores – better solutions will be more likely to be chosen.

Step 4: After creating two parents, now we need to reproduce a child. We raffle a number between 1 to number of buses. From one to this number we take the bus paths from parent one, and from this number until number of buses we take bus paths from parent two. Now we have a new bus paths solution (the child) with a combination from the two parents. Because this procedure can create a child that does not meet the constraint that all stations must be visited at least once, we fix the solution in the same manner done in the initial solutions construction (step 1).

Step 5: With a small probability (set to 0.1) we want to insert a mutation – a new single bus path that replaces an existing path in the child solution which was created in the previous step. If we raffle a lower number than the given probability, this mutation is applied; otherwise do nothing.

Step 6: After checking that the new produced child solution upholds with all algorithm's constraints, we insert the created child solution into the new population set.

Step 7: After receiving the changed solution (child), we compute its score and check whether to insert it into the archive. A solution will be added into the archive in case that it is not dominated in all objective function criteria by any other solution already in the archive.

Step 8: We repeat steps 2-7 a number of times equal to the population size, now we have a new population from all children created by the algorithm – we then replace population with the new population.

Steps 2 through 8 are done iteratively until the stopping criteria is met.

# Results

As we explained in the previous sections, we divided our experimental regime into two methods. Primarily, we wanted to evaluate our algorithms' performances under different parameters configurations. Although every algorithm had several parameters, we decided due to the high computation complexity, to fully evaluate only one parameter for each algorithm – the one we hold to be the most important for the algorithm's performance. The parameters that we chose to optimize were:

- Decrease rate for the SA algorithm: this parameter decides by how much to decrease the current temperature, which in return directly effects the probability of an inferior encountered solution to become the current solution – in an attempt to escape from a local minimum and reach a global minimum. The rest of the algorithm's parameters were set as follows: 1) The initial temperature (TI) was set to 300. 2) The number of iterations on each temperature (TL) was set to 150. 3) The temperature threshold to terminate the search process was set to a temperature of 1. 4) The number of changed solutions that get rejected before the search process terminates was set to 1000.

- Improvement limit for the ABC algorithm: this parameter controls the number of trials in improving the current employed bee's solution by the employed bee itself and by the assigned onlooker bees. If no improvement is reached after limit attempts, the current solution of the employed bee is abandoned, and the employed bee constructs a completely new random solution for itself. The rest of the algorithm's parameters were set as follows: 1) The number of employed and onlooker bees was set to 15. 2) The number of total iterations before the search process terminates was set to 1000.

- Number of iterations for the Genetic algorithm: this parameter controls the number of evolutionary iterations done by the algorithm before the whole search process terminates. The rest of the algorithm's parameters were set as follows: 1) The population size was set to 15.

In order to optimize the algorithms' parameters, we used the changing importance of the objective function's criteria method explained in the previous section and aggregated the results for each parameter value over the changing importance of the objective function's criteria and over the very small, small and medium network size experiments. Figure 4 shows the aggregated results of all parameters tested. Eventually, we decided to set the decrease rate of the SA algorithm to be 0.85, the Improvement limit parameter of the ABC algorithm to 50, and the number of iterations of the Genetic algorithm to 500.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Parameter Name** | **Parameter Value** | **Average Score** | **Average Run Time (Min)** |
| Simulated Annealing | Decrease Rate | 0.75 | -3.169 | 4.632 |
| 0.8 | -3.254 | 4.613 |
| **0.85** | **-3.341** | 4.593 |
| 0.9 | -3.141 | **4.523** |
| Artificial Bee Colony | Improvement Limit | 10 | -2.918 | 5.608 |
| 30 | -3.117 | 5.693 |
| **50** | **-3.404** | **5.517** |
| 150 | -3.296 | 5.756 |
| Genetic | Iterations | 100 | -0.449 | **1.935** |
| 300 | -0.525 | 2.997 |
| **500** | **-0.719** | 4.291 |
| 1000 | -0.634 | 7.589 |

**Figure 4. Parameters aggregated results over the 21 importance objective function criteria**

**combinations and over the very small, small and medium network size experiments**

For the decrease rate parameter of the SA algorithm, we chose the value of 0.85, although it arrived in second place in terms of run time, it arrived in first place in terms of average score, whereas the best option in terms of run time (0.9) arrived at the last place in terms of average score.

For the improvement limit parameter of the ABC algorithm, we chose the value of 50 since it achieved the best results in terms of both run time and average score.

For the iterations parameter of the Genetic algorithm, we chose the value of 500 since it achieved the best average score and since the understandable increase in run time as the iterations grow was the smallest.

Having the chosen parameters of all algorithms at hand, we ran each algorithm with each importance combination once with the chosen parameters. We did so for the very small, small and medium network sizes (these results we already had from the parameters optimization process), but we also ran it for the big network size. Figure 5 shows the average scores for each one of the algorithms for every network size.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Very Small Network** | **Small Network** | **Medium Network** | **Big Network** |
| Simulated Annealing | -1.864 | -1.942 | **-6.213** | **-5.634** |
| Artificial Bee Colony | **-2.091** | **-2.061** | -6.061 | -3.067 |
| Genetic Algorithm | -1.284 | -0.303 | -0.569 | 2.198 |

**Figure 5. Aggregated scores for all importance combinations on each network size**

**using the chosen parameters for each algorithm**

These results show that both the SA and the ABC algorithms are far superior to the Genetic algorithm regarding all network sizes. Comparing only the SA and the ABC algorithms to each other, we can observe that the ABC achieved the best results for the very small and the small network size, whereas the SA algorithm achieved the best results for the medium and big network size. For the very small, small and medium network sizes, the results of the SA and the ABC algorithms are relatively close to each other, whereas for the big network size, the SA algorithm is clearly superior over the ABC algorithm.

For the second part of our experimental regime, we used uniform importance for all objective function's criteria, this is the distance, travel time, arrival frequency and connectivity importance – all importance were set to be 0.25. As explained in the previous section, we ran 10 permutations for each network size i.e., very small, small, medium, big and large networks, giving us a total of 50 runs by each one of our three algorithms. Figure 6 shows the aggregated results of all algorithms over the 10 permutations for each network size, particularly we measured the average score, the number of dominating and dominated solutions in the archive, and the average, min and max run time.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Network Size** | **Algorithm** | **Average Score** | **Dominating Solutions** | **Dominated Solutions** | **Average Run Time** | **Min Run Time** | **Max Run Time** |
| Very Small | Simulated Annealing | -0.763 | 2537 | 729 | **0.79** | 0.76 | **0.80** |
| Artificial Bee Colony | **-0.818** | **8077** | **613** | 0.94 | 0.53 | 1.91 |
| Genetic Algorithm | -0.354 | 206 | 9477 | 0.88 | **0.43** | 1.93 |
| Small | Simulated Annealing | **-1.027** | 961 | **508** | 2.00 | 1.83 | **2.26** |
| Artificial Bee Colony | -0.998 | **8898** | 1738 | 2.04 | 1.17 | 4.15 |
| Genetic Algorithm | 0.036 | 783 | 8398 | **1.15** | **0.46** | 3.56 |
| Medium | Simulated Annealing | **-1.925** | 2038 | **343** | 13.49 | 12.77 | **14.30** |
| Artificial Bee Colony | -1.375 | **2615** | 2831 | 14.83 | 8.44 | 25.35 |
| Genetic Algorithm | 2.573 | 793 | 2272 | **4.34** | **2.69** | 14.72 |
| Big | Simulated Annealing | **-1.876** | **3748** | **146** | 41.05 | 37.49 | **47.61** |
| Artificial Bee Colony | 1.051 | 2025 | 10792 | 40.06 | 25.05 | 61.62 |
| Genetic Algorithm | 7.621 | 1044 | 1879 | **21.99** | **12.18** | 65.09 |
| Large | Simulated Annealing | **-2.448** | 16762 | **0** | 343.05 | 323.62 | 365.90 |
| Artificial Bee Colony | 5.345 | **17320** | 17171 | 183.08 | 140.45 | 238.01 |
| Genetic Algorithm | 12.216 | 839 | 17720 | **54.17** | **42.70** | **86.35** |

**Figure 6. Aggregated results for all network sizes using the archive and uniform**

**importance of objective function's importance method**

Observing the results presented in figure 6, we can state that the Genetic algorithm is superior in terms of run time in all network sizes except for the very small network, however it is extremely inferior to both of the other algorithms under the rest of the evaluation measures and in all network sizes. Except for the average run time of the algorithms, we also looked at the stability of the algorithms in terms of run time, this is measured by the difference between the minimum and the maximum run time. In this run time stability measure, it is well observed that the SA algorithm is far superior to the other two algorithms.

Regarding the average score measure, the SA algorithm achieves the best results in all network sizes except for the very small network size, where the ABC algorithm achieves the best results. The superiority of the SA algorithm in this criterion grows as the size of the network grows.

Regarding dominating and dominated solutions, we obtained these results by comparing each solution in every algorithm's archive to each solution in the other two algorithms' archives. For each such comparison, we checked if one of the two compared solutions dominates the other, if so, the algorithm that the dominating solution came from received a "good" dominating solution point and the algorithm that the dominated solution came from received a "bad" dominated solution point. In terms of dominating solutions, the ABC algorithm has the upper hand in all network sizes except for the big network size where the SA algorithm has the most dominating solutions. Finally, we can see that the SA algorithm has the least number of dominated solutions in all network sizes except for the very small network size.

To sum up this part, it is not easy to come to a clear-cut conclusion about which algorithm is performing best. We can clearly say that the Genetic algorithm is the fastest and is in last place regarding all other measures. However, it is not very clear whether the SA algorithm is superior to the ABC algorithm or the other way around, since the SA algorithm wins most of the time in average score, dominated solutions and stability of run time, but the ABC algorithm wins in most cases in the dominating solutions measure.

We wanted to get a more clear-cut conclusion regarding the SA and the ABC algorithms and suspected that the Genetic algorithm is masking the results between the two with respect to the dominating and dominated solutions measures. Therefore, we excluded the Genetic algorithm from the analysis and evaluated only the SA and ABC algorithm against each other. Figure 7 shows this analysis results, it should be noted that the only change in the results of the two algorithms from Figure 6 is in the dominating and dominated solutions measures.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Network Size** | **Algorithm** | **Average Score** | **Dominating Solutions** | **Dominated Solutions** | **Average Run Time** | **Min Run Time** | **Max Run Time** |
| Very Small | Simulated Annealing | -0.763 | **119** | **74** | **0.79** | 0.76 | **0.80** |
| Artificial Bee Colony | **-0.818** | 74 | 119 | 0.94 | **0.53** | 1.91 |
| Small | Simulated Annealing | **-1.027** | **211** | **88** | **2.00** | 1.83 | **2.26** |
| Artificial Bee Colony | -0.998 | 88 | 211 | 2.04 | **1.17** | 4.15 |
| Medium | Simulated Annealing | **-1.925** | **577** | **81** | **13.49** | 12.77 | **14.30** |
| Artificial Bee Colony | -1.375 | 81 | 577 | 14.83 | **8.44** | 25.35 |
| Big | Simulated Annealing | **-1.876** | **1131** | **23** | 41.05 | 37.49 | **47.61** |
| Artificial Bee Colony | 1.051 | 23 | 1131 | **40.06** | **25.05** | 61.62 |
| Large | Simulated Annealing | **-2.448** | **2317** | **0** | 343.05 | 323.62 | 365.90 |
| Artificial Bee Colony | 5.345 | 0 | 2317 | **183.08** | **140.45** | **238.01** |

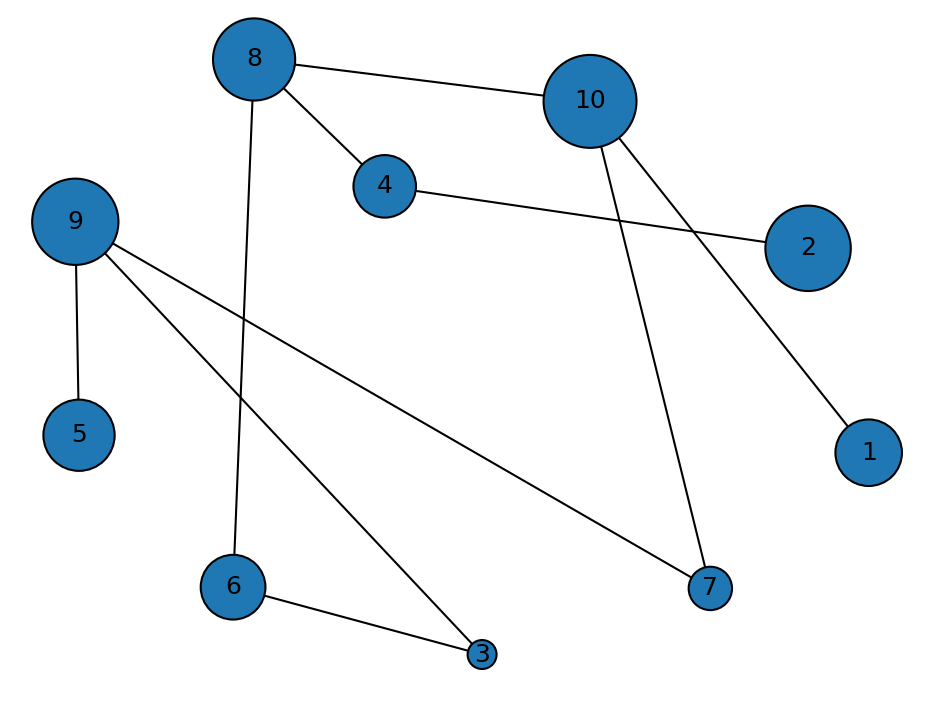
**Figure 7. SA and ABC algorithms results analysis, Genetic algorithm excluded**

Looking at the results at figure 7, it is now easier to declare that the SA algorithm is superior to the ABC algorithm as well. The superiority that the ABC algorithm achieved in the number of dominating solutions when the Genetic algorithm was included in the analysis is now gone and the tables are turned – the SA algorithm has more dominating solutions and less dominated solutions than the ABC algorithm. This finding is explained by that most of the ABC dominating solutions came from dominating Genetic algorithm solutions and not by dominating many SA solutions. As stated before, the SA algorithm is also superior to the ABC algorithm in the average score measure (except for the very small network size), in the dominated solution measure and in the stability of the run time. In terms of average run time, we can conclude that the SA algorithm is quicker in the smaller network sizes, whereas the ABC algorithm is quicker in the bigger network sizes.

In figure 8 one of the very small networks constructed by the SA algorithm can be observed. The network has 3 bus paths – first bus path is colored in red, the second in blue and the third in black. Then nodes' sizes represent their passengers' arrival frequency weight, the bigger the node is, the more frequently passengers arrive at that station. Each edge has two weights – the distance (noted as *d*) and the travel time (noted as *t*), both weights are presented on every edge (transition of a certain bus). The connectivity score of this network is 0 since every node (station) can be reached by the bus paths from any other node in the network.

d=0.37

t=0.09



d=0.01

t=0.58

d=0.43

t=0.53

d=0.16

t=0.13

d=0.16

t=0.15

d=0.09

t=0.03

d=0.22

t=0.07

d=0.53

t=0.04

d=0.12

t=0.15

d=0.28

t=0.37

**Figure 8. A very small network of 10 stations with the bus paths constructed by the SA algorithm.**

**Node sizes correspond to the passengers' arrival frequency and both the distance**

**and the travel time weights are shown on each edge.**

# Conclusion + Future Work

In this paper, we tackled the problem of public transportation traffic network design. This problem is very important to be planned correctly and effectively for the well-being of citizens, but it is also a very difficult task to do so. The complexity of this problem arrives from the multi-objective optimization manner in which it needs to be solved and also due to the enormous search space.

In our setting we took four objective function's criteria i.e., distance and travel time minimization and bus arrival to frequently visited stations and network connectivity maximization.

We implemented three known algorithms and evaluated which of them is the best choice when tackling problems such as this.

The evaluation of these algorithms was done using five experiments that tested them in different network sizes, all networks and weights were synthetically sampled by us.

We optimized one parameter for each algorithm and proceeded with the chosen parameters to evaluate the algorithms with 10 permutations for each network size. We compared the algorithms' average run time, stability of run time, average weighted objective function score, dominating and dominated solutions over the others. This procedure was done using a uniform importance for all objective function's criteria. However, we allow the user to input his own subjective importance and get the best results that correspond to these importance weights.

Our analysis has shown that in general the Simulated Annealing algorithm was the best performing algorithm, achieving superiority in run time stability, average objective function score, dominating and dominated solutions. The Genetic algorithm was the fastest out of the three, however it was far inferior in all other criteria evaluated. The Artificial Bee Colony algorithm was evaluated by us as the second-best algorithm, achieving very similar results to the SA algorithm in the small network sizes and being quicker than the SA algorithm in the larger network sizes.

For future research, we would suggest obtaining data from some municipality about stations locations, passengers' frequency of arrival to those stations, travel time and distance from each station to all others, this way our suggested algorithms can also be evaluated in a real-life scenario. In addition, we would like to see the effect of time limitation on the algorithms, meaning how will results be affected by different time limitations. This is important since for large networks the suggested algorithms have a relatively long run time, thus setting a time limitation that will not affect results dramatically will be a valued asset. Other parameters of the algorithms can also be optimized in future work.

# References

[1] A. Migdalas, “Bilevel programming in traffic planning: models, methods and challenge,” Journal of Global Optimization, vol. 7, no. 4, pp. 381–405, 1995.

[2] H. Yang and M. G. H. Bell, “Models and algorithms for road network design: a review and some new developments,” Transport Reviews, vol. 18, no. 3, pp. 257–278, 1998.

[3] K. Kepaptsoglou and M. Karlaftis, “Transit route network design problem: Review,” Journal of Transportation Engineering, vol. 135, no. 8, pp. 491–505, 2009.

[4] R. Z. Farahani, E. Miandoabchi, W. Y. Szeto, and H. Rashidi, “A review of urban transportation network design problems,” European Journal of Operational Research, vol. 229, no. 2, pp. 281–302, 2013.

[5] X. Xu, A. Chen, and C. Yang, “A review of sustainable network design for road networks,” KSCE Journal of Civil Engineering, vol. 20, no. 3, pp. 1084–1098, 2016.

[6] Nayak, S. (2020). Multiobjective optimization. In Fundamentals of Optimization Techniques with Algorithms (pp. 253–270). Elsevier.

[7] Givi, M. K. B., & Asadi, P. (2014). Advances in Friction-Stir Welding and Processing. Advances in Friction-Stir Welding and Processing (pp. 1–756). Elsevier Ltd.

[8] Melanie, M. (1996). An introduction to genetic algorithms By Melanie Mitchell. MIT Press, Cambridge, MA. (1996). 205 pages. $30.00. Computers & Mathematics with Applications, 32(6), 133.

[9] Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. Applied soft computing, 8(1), 687-697.‏

# Appendix A. All objective function's criteria importance possible combinations

|  |  |  |  |
| --- | --- | --- | --- |
| **Distance Importance** | **Travel Time Importance** | **Passengers' Arrival Importance** | **Connectivity Importance** |
| 0.7 | 0.1 | 0.1 | 0.1 |
| 0.1 | 0.7 | 0.1 | 0.1 |
| 0.1 | 0.1 | 0.7 | 0.1 |
| 0.1 | 0.1 | 0.1 | 0.7 |
| 0.5 | 0.3 | 0.1 | 0.1 |
| 0.5 | 0.1 | 0.3 | 0.1 |
| 0.5 | 0.1 | 0.1 | 0.3 |
| 0.3 | 0.5 | 0.1 | 0.1 |
| 0.1 | 0.5 | 0.3 | 0.1 |
| 0.1 | 0.5 | 0.1 | 0.3 |
| 0.3 | 0.1 | 0.5 | 0.1 |
| 0.1 | 0.3 | 0.5 | 0.1 |
| 0.1 | 0.1 | 0.5 | 0.3 |
| 0.3 | 0.1 | 0.1 | 0.5 |
| 0.1 | 0.3 | 0.1 | 0.5 |
| 0.1 | 0.1 | 0.3 | 0.5 |
| 0.3 | 0.3 | 0.3 | 0.1 |
| 0.3 | 0.3 | 0.1 | 0.3 |
| 0.3 | 0.1 | 0.3 | 0.3 |
| 0.1 | 0.3 | 0.3 | 0.3 |
| 0.25 | 0.25 | 0.25 | 0.25 |