Object-Oriented Research Framework for the Fireworks Algorithm with the Focus on the Travelling Salesman Problem

Robert Ehni and Carsten Müller

***Abstract*—The fireworks algorithm is a lately developed algorithm based on fireworks in the night sky. It is a swarm intelligence algorithm with a broad range of use. In this paper the algorithm is introduced and discussed. Furthermore, an implementation of the algorithm for optimizing the travelling salesman problem is introduced. Prior to implementation the adjustments to the algorithm will be provided. The implementation is based on Java and the layer based and extendable architecture of the implemented application is introduced. The performance of the implementation and the impact of the parameters on the behavior of the algorithm will be tested and analyzed. The results show that the fireworks algorithm is a efficient and performant algorithm to optimize the combinatorial problem of the travelling salesman.**

***Index Terms*—Travelling salesman problem, combinatorial problem optimization, fireworks algorithm.**

I. Introduction

The traveling salesman problem (TSP) is a well-known combinatorial problem. Although the idea behind the problem may not sound too complex, the TSP is representing a difficult combinatorial optimization problem. The goal is to find the minimum route over a given set of cities while the starting location is also the ending location and each city is visited once [1].

In this paper a java-based implementation of the fireworks algorithm (FWA) framework for optimizing the TSP will be introduced together with the needed foundations. The fireworks algorithm is a recently developed algorithm. It is a swarm intelligence algorithm based on the explosions of fireworks in the night sky. Swarm intelligence has attracted interest in researchers globally in the recent years [2]-[4].

The paper is organized as follows. In Section II, the foundations of the TSP and the fireworks algorithm are introduced. The discrete fireworks algorithm for the TSP is presented in Section III. Section IV describes the software architecture of the implemented application. The proof of concept for the application and results of the parameter analysis are given in details in Section V.

Finally, the conclusion summarizes in final Section VI.

Manuscript received July 15, 2018; revised October 8, 2018.

Robert Ehni is with the Department of Applied Informatics, Swarm Intelligence Research, Baden-Wuerttemberg Cooperative State University Mosbach, Lohrtalweg 10, 74821 Mosbach, Germany (e-mail: rob.ehni.15@lehre.mosbach.dhbw.de).

Carsten Müller is with the Department of Information Technologies, University of Economics, Faculty of Informatics and Statistics, W. Churchill Sq. 4, 130 67 Prague 3, Czech Republic (e-mail: research@ieoca.org).

II. Foundations

The foundations for this paper are the travelling salesman problem together with the fireworks algorithm. Within this chapter the foundations for these two parts will be provided. In the following chapters the foundations will be used to implement the software architecture.

A. The Travelling Salesman Problem

For this paper the following definition of the travelling salesman problem will be used within this paper: Given a set of N cities $\{c_1, c_2, c_3, ..., c_n\}$ the distance between two cities $c_a c_b$ is described with $d(c_a, c_b)$ whereas for the optimization of the problem a permutation x of the cities N is to be found with a minimum tour length. In this tour all the cities must be visited exactly once, and the beginning of the tour must also be the ending point of the tour. The tour length of the permutation x with the last city as c_{x_n} is defined with

$$L(x) = \sum_{i=1}^{N-1} d(c_{x_i}, c_{x_{i+1}}) + d(c_{x_n}, c_{x_1}) \quad (1)$$

Furthermore, this paper only uses symmetrical problems, so that the distance between two cities is the same for all directions [1].

B. Foundations of the Fireworks Algorithm

This subsection will describe the overall working mechanism and the foundations of the fireworks algorithm and its four core components: the explosion operation, the mutation operation, the mapping rule and the selection strategy. In the next chapter III a detailed view into the working mechanism of the implemented algorithm will be given.

The fireworks algorithm introduced by Y. Tan originates from, like the name already announces, fireworks in the night sky. Based on the behavior of the explosions from the fireworks the way to search the solution space will be adapted in this algorithm [5].

The algorithm is working in an iterative way like evolutional algorithms where each iteration is using information from the previous one. In each iteration all four components of the fireworks algorithm are applied to the problem. The algorithm will run until a defined ending criteria is met; either the maximum number of iterations or the accuracy requirement of the problem. The components within the fireworks algorithm are designed to adopt the behavior of the real counterparts: the explosion operation, the

DOI: 10.7763/IJCTE.2018.V10.1215

mutation operation, the mapping rule and the selection strategy [6], [7].

Fig. 1 briefly shows the basic process of the algorithm.

```
Initialize population
while Ending criteria not fulfilled do
   Explosion operation
   Mutation operation
   Mapping rule
   Selection strategy
end while
```

Fig. 1. Pseudocode for the fireworks algorithm. In an iterative way each component of the algorithm is applied until an ending criteria is met. Own illustration.

In the following the four components of the algorithm and their purpose will be described [5], [8]-[11]:

1) Explosion operation

The first component of the fireworks algorithm is the explosion operation. Its main purpose is to generate new sparks from the existent fireworks. The component contains three parts: the explosion strength, the amplitude and the displacement operation. The explosions strength controls, how many sparks are generated by each firework. For a efficient exploration of the feasible space fireworks with a good fitness value (good explosions) will generate more sparks than fireworks with worse fitness values (bad explosions). The amplitude will control the radius of the explosion and therefore enhance the local or global search aspect of the explosion. A good explosion will have a higher amplitude and thus focus on a local search to improve the fitness value while a bad explosion will focus on the global space. The last part of the explosion operation is the displacement operation. For each firework the amplitude will be randomly adjusted with the displacement operation. This is done to ensure a diversity of the fireworks.

2) Mutation operation

After the explosion operation the mutation operation takes place to furthermore improve the diversity of the population. The mutation operation will pick a (e.g. random) number of fireworks and apply a mutation operation. What kind of mutation operation will be applied to the fireworks is dependent on the problem which should be optimized.

3) Mapping rule

If there is a feasible space for the problem and a firework is near the boundary of this space some of the generated sparks of the firework could be out of the feasible space. These generated sparks may be useless and thus the mapping rule takes place to bring them back into the feasible space. With this operation we guarantee that every solution is in the feasible space.

4) Selection strategy

The last step in each iteration is the selection strategy. This operation selects the sparks which will be used for the next iteration. The best spark will always be selected, for the rest of the sparks the selection strategy will select them on a random and distance-based method. This is done to ensure a diversity of the fireworks. Thus, sparks with a greater distance to other sparks will have a higher probability of being selected for the next iteration.

III. Object-Oriented Research Framework

The core fireworks algorithm described in II.B needs to be customized to optimize the travelling salesman problem. The behavior of the algorithm stays mainly the same as seen in Fig. 2. Major changes are the introduction of the two explosion operators. The amplitude of the explosion operation will also made to a global parameter throughout the algorithm. The mutation operation will be used for further exploration of the feasible space and the mapping rule becomes obsolete due to the nature of the problem and the explosion and mutation operation. All these changes will be described in the following. Furthermore, a detailed view into the mechanisms in each component will be given [5].

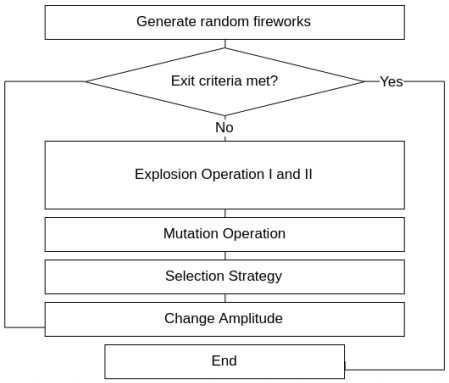

Fig. 2. Flowchart for the discrete fireworks algorithm for optimizing the travelling salesman problem. Own illustration after [5].

To demonstrate the behavior of the operations and their relationships a numerical example will be used throughout this chapter. Given a set of six cities with index and coordinates: 1 (1/1), 2 (4/1), 3 (8/2), 4 (6/7), 5 (2/6), 6 (3/4). A random initial route which should be optimized is set as 1 – 6 – 3 – 4 – 2 – 5 – 1 with an overall distance of 31, 18 length units (see Fig. 3).

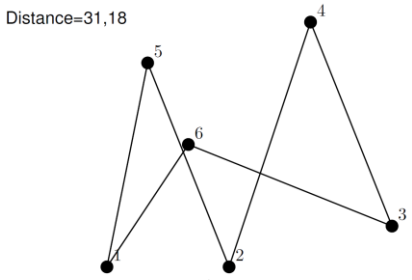

Fig. 3. Numerical example instance of the travelling salesman problem with six cities. This instance with the shown initial route is used in this paper for demonstrating the behavior of the components of the fireworks algorithm.

A. Explosion Operation

The explosion operation undergoes major changes to optimize the travelling salesman problem. One major change is the function and behavior of the amplitude. In the core fireworks algorithm the amplitude is calculated for each firework depending on the fitness value. It controls the behavior of the searching function of the explosion operation in terms of local or global search. In the discrete algorithm this variable will have the same function, but it will be a global parameter for all fireworks. The amplitude will be adjusted after each iteration. To avoid getting stuck in a local optimum the algorithm will adjust the amplitude after a fixed

number of iterations (which we will call β) without a decreasing distance to enhance the global search functionality. The explosion operation from the fireworks algorithm will be split into two operations in the discrete algorithm: explosion operation I and II. The two operation explosions are based upon optimization algorithms for the travelling salesman problem. Before each operation the number of sparks for each firework, i.e. the explosion strength, is calculated with

$$sparks_{x^i} = M_{\text{sparks}} * x^i_{\text{normalized}} \quad (2)$$

where x^i is the current firework, the constant M_{sparks} is the maximum number of sparks per firework and $x^i_{\text{normalized}}$ is a normalized value based on the distance of the current firework between 0 and 1. The better the fitness, i.e. the distance of the trip, the better the value. The best trip will have $x^i_{\text{normalized}}$ set as 1 and the worst as 0. The normalization is done with

$$x^i_{\text{normalized}} = 1 - \frac{L_{x^i} - L_{\min}}{L_{\max} - L_{\min}} \quad (3)$$

where L_{x^i} is the distance of the current firework and $L_{\max}$ and $L_{\min}$ are the worst and best distance in the current population. [5]

Each explosion operation is based on an optimization method and tries to generate sparks with a better fitness value. Although sparks with a worse fitness value will be discarded there is a chance that a worse spark will be accepted for the next operations. The probability for accepting a worse spark is calculated with

$$p_a = e^{\frac{-L_m}{L_o} * \alpha}. \quad (4)$$

where p_a is the probability, L_o is the original fitness of the spark, L_{m} the mutated fitness of the newly generated spark and α is the global amplitude. A closer distance between L_{m} and L_o will increase the probability of accepting the worse solution.

1) Explosion operation I

Explosion operation I is based on the two-opt optimization method for the travelling salesman problem which takes two edges in the route and swaps them. It is a post optimization method which goes over each city and selects the cities $a = random, \quad b = a+1, \quad c = currentCity, \quad d = c+1$ whereas it skips the intersections and calculates the original distance $L_o = d(a,b) + d(c,d)$ and the mutated distance $L_m = d(a,c) + d(b,d)$. If L_{m} is shorter the changes will be applied to the firework and thus a new spark is generated [12].

If a worse solution gets accepted during the operation an additional two-opt optimization will be applied to the worse solution to decrease the probability of missing a potential solution in the local space [5].

Fig. 4 shows a possible result of the first explosion operation applied to the initial example route. The edges 6 – 3 and 2 – 5 are swapped with 6 – 2 and 3 – 5 which results in a shorter distance and thus the new generated trip will be accepted.

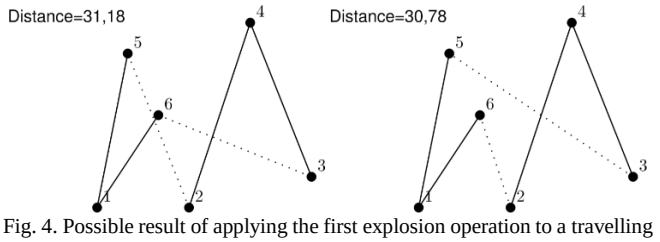

Fig. 4. Possible result of applying the first explosion operation to a travelling salesman problem. Left: initial example. Right: route after explosion. The affected edges are dotted.

2) Explosion operation II

Explosion operation II is based on the three-opt optimization method [13] and is working in a similar way like the first explosion operation. Although in this operation three edges are swapped instead of two. Cities a, b, c, d, e, f are selected (similar to explosion operation I) and $L_o = d(a,b) + d(c,d) + d(e,f)$ and the possible mutations L_{m} (e.g. $L_m = d(a,d) + d(e,b) + d(c,f)$) are calculated. If one of the new mutations are shorter, a new spark will be generated. Like in the first explosion operation, if a worse solution will be accepted it will be immediately further [5].

While explosion operation I is only swapping two edges, it may miss out some routes which can only be created when swapping more edges at a time. Thus, the explosion operation II is introduced to gain diversity and enhance the searchable space.

Fig. 5 demonstrates the working behavior of the explosion operation II applied to the initial route. The edges 1 – 6, 3 – 4 and 2 – 5 are replaced by the new edges 1 – 4, 2 – 6 and 3 – 5. The new route is now longer and will only be accepted when p_a is met.

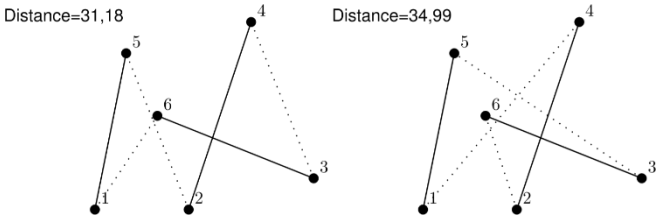

Fig. 5. Possible result (right) of the explosion operation II applied to the initial example route (left). Affected edges are dotted. Although the new route is now longer it may be accepted if the probability p_a is met.

B. Mutation Operation

For an optimal exploration of the feasible space and to further maintain diversity the mutation operation is used in addition to the explosion operations. During the operation a loop will iterate over each city i in the route and another random city z will be selected. If there is no direct connection between these two cities the city z will be set in the place between i and $i+1$ so that the route is now

$\{...,c_i,c_z,c_{z+1},...\}$. When a shorter route is found the changes will be applied to the spark [5].

Because of the optimization methods used in the explosion operations only multiple edges at once are swapped. Thus, some routes can be hard to optimize. Therefore, the mutation operation comes in place to fill this gap. Fig. 6 shows a possible result of the mutation operation where city 6 moved in between city 5 and 2 to create a new route which couldn't be created by the explosion operations.

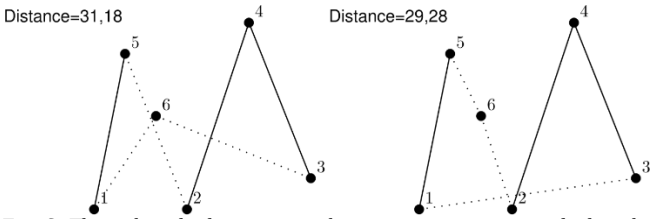

Fig. 6. The right side demonstrates the mutation operation applied to the initial example route on the left side. City 6 is taken from its original position and moved in between city 5 and 2. The edges affected by this operation are dotted.

C. Mapping Rule

The purpose of the mapping rule was to take solutions out of the feasible space and map them back into the feasible space. Due to the nature of the explosion operations and the mutation operation it is impossible to create a solution out of the feasible space. Thus, there is no need of mapping rule anymore [5].

D. Selection Strategy

For the discrete fireworks algorithm we adopt the behavior of the core algorithm. The best spark is always kept for the next iteration. For the rest we normalize the distances (eq. (3)) and select randomly based on the normalized distances. So, sparks with a better fitness value have a better chance to get selected for the next iteration.

E. Parameters of the Discrete Fireworks Algorithm

For the discrete fireworks algorithm we have multiple parameters which control the behavior of the algorithm.

1) Explosion amplitude α

The explosion amplitude α controls the search behavior. A high amplitude will enhance local search while a low amplitude will enhance the global search behavior by accepting worse solutions in the explosion operations. In the beginning the explosion amplitude is set to a neutral value and will be automatically adjusted by the algorithm during runtime.

2) Population size

Sets the number of fireworks, i.e. trips, which will be generated and used for optimization.

3) Maximum amount of sparks per explosion

Sets the maximum amount of sparks a firework can generate per explosion.

4) Factor β

After each iteration the amplitude will be adjusted to either improve local or global search. When the algorithm doesn't find a better solution after a given number of iterations, it will automatically enhance the global search behavior to further explore the feasible space. This factor sets the number of iterations, after which the global search will be increased.

5) Minimum distance

This is an ending criteria for the algorithm. When the given minimum distance is met by one firework the algorithm will stop.

6) Number of iterations

Next to the minimum distance, the number of iterations is the second ending criteria of the algorithm. When the minimum distance is not met the algorithm will stop after a given number of iterations.

IV. Software Architecture

The software architecture of the implemented application is split into three layers: the presentation, application and persistence layer.

Each layer is independent of each other. The application layer contains the core component of the application: the fireworks algorithm as introduced in Section III. Furthermore, the application layer contains the logic for the TSP. The persistence layer has three main responsibilities: reading in TSPs from text files and providing the information to the application layer, reading and writing the parameter configuration for the fireworks algorithm and reading and writing the data from the algorithm which is produced during runtime. Fig. 7 shows a visualization of the three layers and their responsibilities.

The layers will be described in the following.

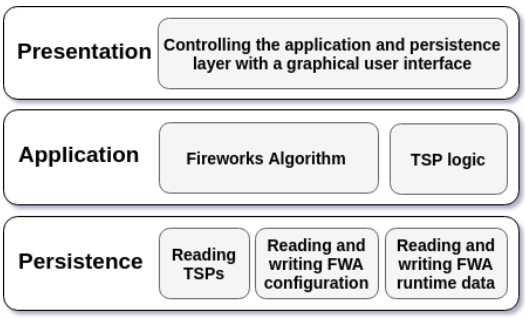

Fig. 7. Overview of the software layers and their responsibilities.

A. Presentation Layer

The presentation layer is implemented using JavaFX to provide a graphical user interface to control the application layer. Within the user interface, the user can load a TSP from a text file. After successfully loading a text file the parameters for the algorithm (see Section III.E) can be adjusted. Finally, the algorithm can be started. During runtime the graphical user interface provides the runtime data from the algorithm and an option to stop the execution.

B. Persistence Layer

The persistence layer provides three major functionalities throughout the application: Reading the given tsp problem as a text-file, reading and writing the parameters for the algorithm and storing the run time algorithm data. To store the data the in-memory database HSQLDB is used. This allows the application layer to subsequently access the runtime data afterwards to export the information for analysis.

Fig. 8 shows the class diagram for the three main classes for the persistence layer: the Configuration, the HSQLDBManager and the TSPReader.

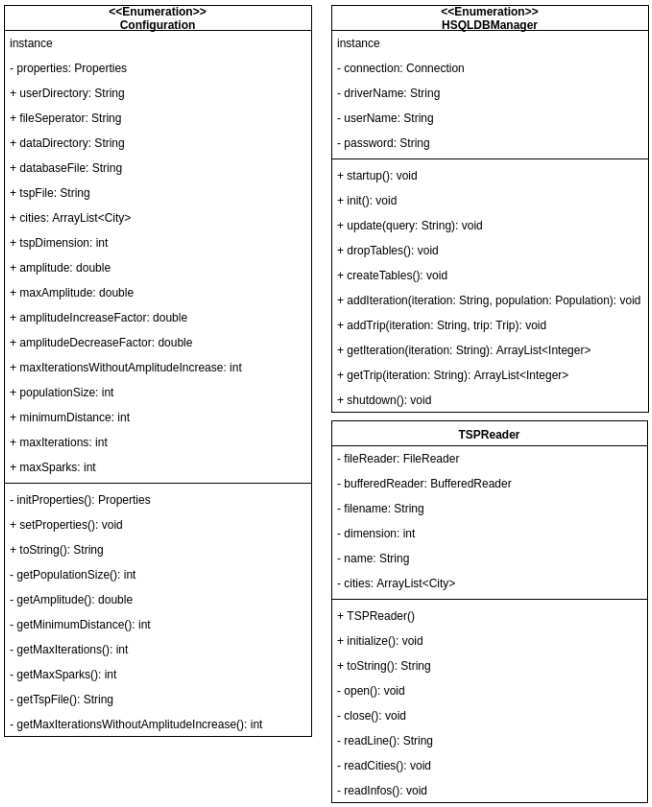

Fig. 8. Class diagram of the fireworks algorithm components.

C. Application Layer

The application layer is the main part of the software architecture. It contains multiple parts: the implementation of the tsp, the central mediator for controlling the algorithm and the implementation of the fireworks algorithm as described in Section III whereas the algorithm and the tsp implementation are the core of the application layer.

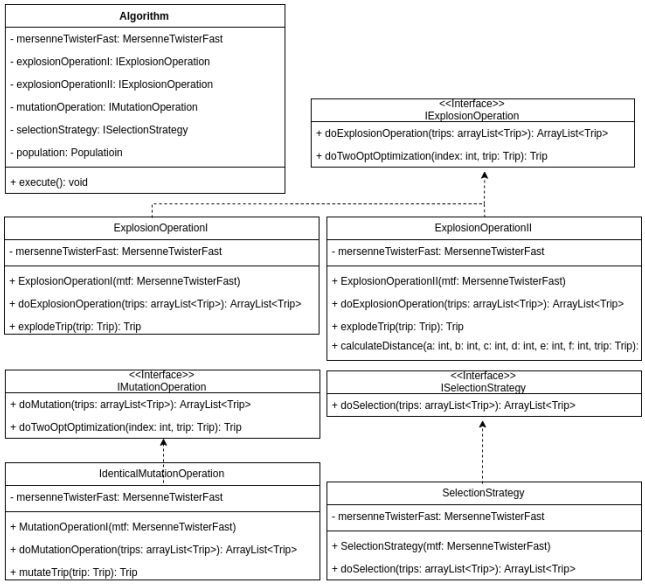

Fig. 9. Class diagram of the fireworks algorithm components and the algorithm controller.

The fireworks algorithm is separated in the three components of the discrete algorithm controlled by single class: the explosion operation, the mutation operation and the selection strategy. Each component is implemented independently of each other and accessed by defined interfaces. The main class is implemented for controlling the algorithm and accessing each component. Thus, an exchange or modification of each component can be done easily. Fig. 9 shows the mediator class for the algorithm the class diagrams for the interfaces of the components and their implementation.

After first tests of the implemented algorithm a huge bottleneck in the explosion and mutation components was found due to the sequential implementation in the beginning. Due to this both components have been overhauled. By using Java Streams a great amount of parallelism was achieved within the components. This improvement increased the overall performance of the algorithm by more than 50%.

V. Experimental Results and Analysis

In the first part of this section the parameter impact on the algorithm will be discussed and analyzed. The second part will focus on the results of the test runs of the parameter analysis and the overall performance of the implemented application. For the parameter analysis twelve individual scenarios (see TABLE for the parameter setup and Fig. 10 for the boxplots of each scenario result) are run on the implemented application against the TSP a280. The current optimum for the TSP a280 is 2579.

TABLE I: Tested Scenarios for the Proof of Concept. All Scenarios Are Run Against the TSP A280

Scenario #	Max. Sparks	Population Size	Factor β	Number of Iterations
#1	1	5	10	
#2	3	5	10	
#3	5	5	10	
#4	10	5	10	
#5	15	5	10	
#6	20	5	10	20.000
#7	10	10	10	
#8	15	10	10	
#9	20	10	10	
#10	10	10	15	
#11	15	10	15	
#12	20	10	15	

All the scenarios are run on an Intel® Core™ i5-6300U CPU with 8 GB RAM running Windows 10 64bit. For all scenarios the maximum number of iterations is set to 20.000 iterations. The maximum number of sparks per explosion will be tested with the values 10, 15 and 20 with a population size with 5 and 10 each plus an additional three scenarios with the maximum sparks of 1, 3 and 5. Factor β is set to 10 for all the scenarios except the last three. For these scenarios the impact of a higher factor β on the behavior of the algorithm is analyzed.

The longer upper whiskers in the boxplots from Fig. 10 show that almost all scenarios are quickly heading towards a suboptimum during the optimization process. The shorter lower whisker on most of the scenarios demonstrates the good overall performance of staying close to the suboptimum. As seen in Fig. 11 the runtime is affected from both the

population size and the maximum number of sparks. Both parameters affect the runtime in a linear manner. While the last two scenarios are not affected by the higher factor β scenario #10 is an outlier with an 50% longer runtime compared to scenario #7 with the same parameters except the lower β.

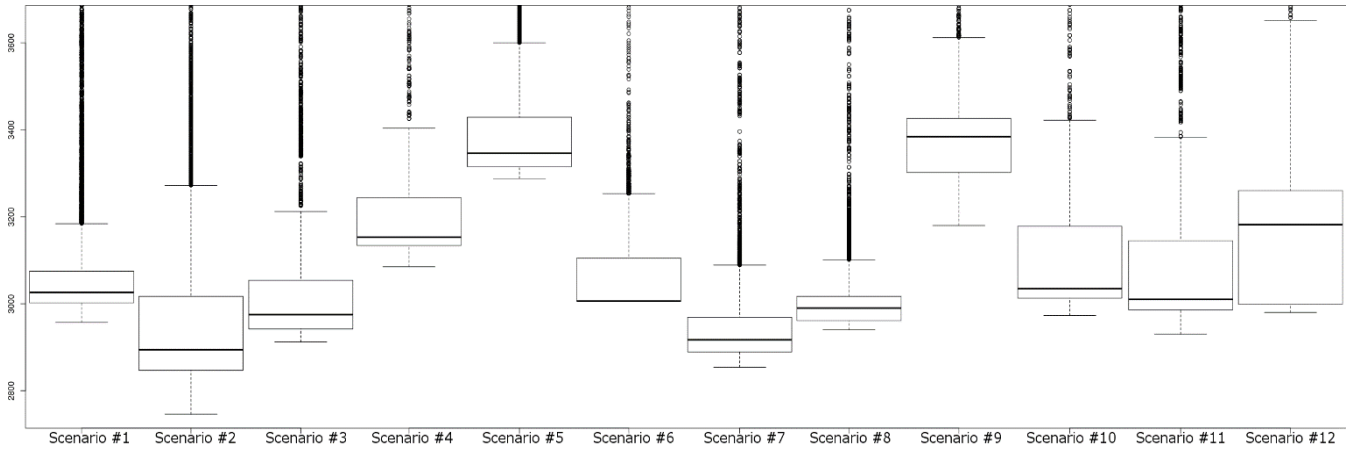

Fig. 10. Boxplot overview within the range of 2700 up to 3700 for each test scenario. Some significant information shown by the boxplot for example is the direct influence of the Factor β in scenario #10, #11 and #12 compared to #7, #8 and #9 which is represented in a drastically increased range of relevant results. Also shown are the good drilling capabilities of the algorithm by using small parameters for the maximum number of sparks and the population size in the first three scenarios.

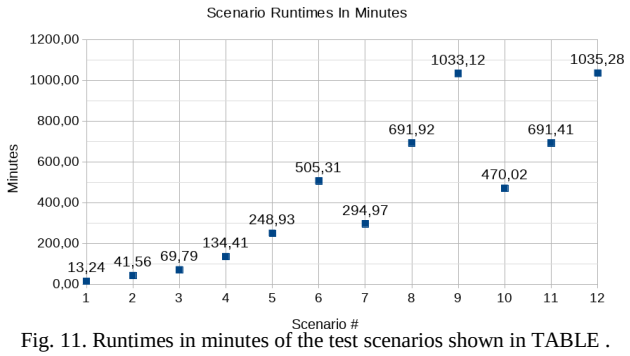

Fig. 11. Runtimes in minutes of the test scenarios shown in TABLE .

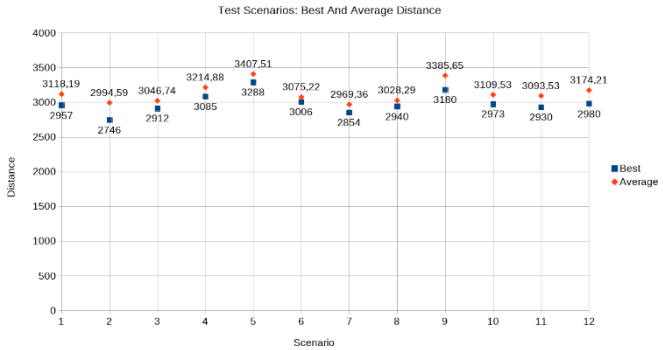

Fig. 12. Best and average distance of the test scenarios shown in TABLE .

Fig. 12 shows the best and average distance of each test scenario. Even with the high amount of parallelism the performance of the algorithm reduces with a growing population or spark number while giving no major improvements of the produced solution quality. Scenario #2 provides the optimal results of all scenarios with a best distance of 2746 which is equal to a solution quality of 93.25% for this TSP instance. With a runtime of 41 minutes this demonstrates the drilling capability of the algorithm with small parameters of a population of 5 and a maximum amount of 3 sparks per explosion operation.

VI. Conclusion

This paper proposes an implementation of the fireworks algorithm for the TSP. Based on the fireworks algorithm proper changes are introduced to optimize the TSP.

The explosion operation is split into two operations. The purpose of the mutation operation becomes to support the explosion operation and the mapping rule is not required in this case. A major advantage of the implemented fireworks algorithm is the component-based architecture.

An introduction of new mechanics for the single components can be done with a minimum amount of changes to the overall algorithm. For example, in addition to the two-opt and three-opt optimization methods implemented with the explosion operation I and II some additional optimization methods for the TSP could be introduced alongside.

The performance of the implementation provides a proof of concept by finding a suboptimum of 2746. The quality of the found suboptimum is equal to 93.25% based on the current optimum of 2579 for the TSP a280. Though the provided implementation still can be optimized, this work shows the efficiency and performance of the fireworks algorithm.

References

[1] E. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, and D. Shmoys, *The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization*, East Kilbrite: Courier International Limited, 1992.

[2] S. Garnier, J. Gautrais, and G. Theraulaz, "The biological principles of swarm intelligence," *Swarm Intelligence*, 2007.

[3] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 760-766.

[4] P. E. Andries and P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[5] Y. Tan, *Fireworks Algorithm - A Novel Swarm Intelligence Optimization Method*, Heidelberg: Springer-Verlag Berlin Heidelberg, 2015.

[6] X. G. Li, S. F. Han, and C. Q. Gong, "Analysis and improvement of fireworks algorithm," *Algorithms*, 2017.

[7] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," *Advances in Swarm Intelligence*, 2010.

[8] Y. Tan, Y. Chao, S. Zheng, and K. Ding, "Introduction to fireworks algorithm," *International Journal of Swarm Intelligence Research (IJSIR)*, 2013.

[9] J. Li, S. Zheng, and Y. Tan, "Adaptive fireworks algorithm," in *Proc. the 2014 IEEE Congress on Evolutionary Computation (CEC)*, Jul. 2014, pp. 3214-3221.
[10] S. Zheng, A. Janecek, and Y. Tan, "Enhanced fireworks algorithm," in *Proc. the 2013 IEEE Congress on Evolutionary Computation (CEC)*, 2013.
[11] J. Liu, S. Zheng, and Y. Tan, "The improvement on controlling exploration and exploitation of firework algorithm," *Advances in Swarm Intelligence*, 2013.
[12] G. Croes, "A method for solving traveling salesman problems," *Operations Research*, 1958.
[13] S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, 1965.

Robert Ehni was born on the 21 October, 1992 in Sinsheim, Germany. He is currently studying applied computer science at the Cooperative State University Mosbach where he will graduate with a bachelor of science in September 2018.

Before he started studying he did an apprenticeship to become an IT businessman after which he worked as an IT systems administrator at the City Administration of Sinsheim.

During the apprenticeship he discovered his passion for computer science and software development and thus he attained his advanced technical college entrance qualification to start studying applied computer science.