Received: *02.03.2022*
Accepted: 06.03.2022

Application Of Chicken Swarm Optimization to Travelling Salesman Problem and A Reviewing Of Similar Studies

Süleyman ÖZER[*1,2], Adil BAYKASOĞLU[3], Özcan KILINÇCI[3]

[1] Dokuz Eylül University, Institute of Natural and Applied Sciences, Industrial Engineering Department, 35930, İzmir, Turkey
[2]KTO Karatay University, Faculty of Engineering and Natural Sciences, Industrial Engineering Department, 42020, Konya, Turkey
[3]Dokuz Eylül University, Faculty of Engineering, Industrial Engineering Department, 60250, 35930, İzmir, Turkey

Abstract

This study focuses on an adapted application of the **C**hicken **S**warm **O**ptimization (CSO) Algorithm on a **T**ravelling **S**alesman **P**roblem (TSP). CSO Algorithm aims to search for optimal solution of a continuous function metaheuristically as a basis and it need some modifications to be coupled to a discontinuous problem like TSP. Some studies have been done before in the process of transforming a continuous metaheuristic method into discontinuous. However, as seen in reference studies, the algorithm needs also an additional decision-making mechanism after the transformation, and this would usually be the **G**reedy **S**earch (GS) Algorithm when it comes to the CSO. Nevertheless, the aftermath of these decision-making mechanisms the customized novel CSO leaves the main logic of CSO and being Swarm Intelligence Algorithm and turn into a more colorful variation of the casual GS algorithm. The original part that distinguishes this work from others, it is focused on applying the CSO algorithm to a discontinuous TSP problem, while staying true to neutral phenomenon mimicked method and preserve the CSO's logical context. The main quest of the paper is not to invent a method that gives better results for the example problem on any account, but to reveal how the CSO algorithm will give results to the example problem if it maintains its logic integrity. Therefore, an extension free bare adaptation of CSO is implemented for a TSP problem and results are observed.

Keywords: Chicken Swarm Optimization, CSO, TSP, Greedy Search, Metaheuristic, Discontinuous Transformation.

Tavuk Sürüsü Optimizasyonunun Gezgin Satıcı Problemi Üzerinde Uygulaması ve Benzer Çalışmaların İncelenmesi

Süleyman ÖZER[*1,2], Adil BAYKASOĞLU[3], Özcan KILINÇCI[3]

Özet

Bu makalede, aslen bir sürekli fonksiyonun en iyi çözümünü metasezgisel olarak aramayı hedefleyen **T**avuk **S**ürüsü **O**ptimizasyonu (TSO) algoritmasının, **G**ezgin **S**atıcı **P**roblemi (GSP) üzerinde bir uygulaması yapılmıştır. TSO, sürekli bir amaç fonksiyonuna sahip olan problemin optimum değerini arayan metasezgisel bir yöntemdir ve TSP gibi amaç fonksiyonu kesikli olan bir probleme akuple edilebilmesi için bir dönüşümden geçirilmesi gerekir. Sürekli fonksiyonlar için tasarlanmış olan TSO mekanizmasının kesikli duruma getirilmesi sürecinde daha önceden başka çalışmalar yapılmıştır ve referans çalışmalarda görüldüğü üzere, böyle bir dönüşümden sonra algoritma ek bir karar mekanizmasına ihtiyaç duymaktadır. TSO özelinde konuşursak bu mekanizma genellikle **G**reedy **S**earch (GS) algoritması olmaktadır. Fakat bu referans çalışmalarda görüldüğü kadarıyla, dönüşüm sonrasındaki böyle bir karar verme mekanizmasında TSO'nun ve Doğa İlhamlı olmanın esas mantığından çıkarak GS algoritmasının daha renkli bir varyasyonu haline dönüşmektedir. Bu çalışmayı diğerlerinden ayıran özgün kısım, TSO algoritmasını kendi mantıksal bağlamından

korpartmadan ve özüne sadık kalarak, örnek bir kesikli probleme (GSP) uygulamaya odaklanmış olmasıdır. Makalenin esas arayışı, örnek problem için daha iyi sonuçlar veren bir TSO dönüşümü keşfetmek değil, TSO algoritmasının bütünlüğünü koruduğu takdirde örnek probleme nasıl sonuçlar vereceğini ortaya çıkartmaktır. Bu nedenle, yalın bir TSO dönüşümü herhangi bir ek mekanizma olmadan örnek bir TSP problemine uygulanmış ve sonuçlar elde edilmiştir.

Anahtar Kelimeler :Tavuk Sürüsü Optimizasyonu, GSP, Greedy Search, Metasezgisel, Kesikli Dönüşüm.

1. Introduction

Neutral phenomenon mimicking algorithms are subset of metaheuristic algorithms which can basically be classified under three groups as Evolutionary, Swarm Intelligence Based and Bio-inspired. It is not hard to predict the working principles of these methods by looking at their names. Evolutionary algorithms design a solution universe based on the creation of genetic copies and biodiversity in the process, where the strong individuals (better fitness values) have a better chance of survival. As to Bio-inspired algorithms, they are methods that take their inspiration from nature and customize it according to their own specifications, rather than completely imitating nature. Swarm Intelligence Based (SIB) algorithms, on the other hand, reap the fruits of the evolution process that has already been filtered for millions of years by mimicking the organization of a social animal colony. The CSO algorithm, which is discussed in this study, is a SIB algorithm that seeks the optimum by imitating the search for food in nature. Colonies consisting of roosters, hens and chicks form the herd are in competition with each other to reach richer food resources.

The explanation of the CSO algorithm is beyond the scope of this study, hence, it would be more appropriate to get the details about the CSO directly from the creators of the idea and algorithm, Meng at al. [1]. Additionally, many variations of the CSO algorithm are also available for different purposes. Enthusiasts can found some in the sub-sections of Deb et Al. [2]. But here it is necessary to mention the parameters related to CSO to avoid possible communication failure: I represent maximum iteration number, k represent sub-swarm number of the swarm, i represent the chicken number of each sub-swarm, G represent the rate of rebuilding hierarchical order, P represent the rate of recombining bad solutions from the swarm, H represents the percentage of Hens in each sub-swarm.

Flow of the paper fallows these steps: In the second chapter of the study, the adaptation of the CSO algorithm to the TSP problem will be examined. The necessity of the decision-making mechanism will be questioned. At the third section, an application will be rebuild and the results will be investigated in the absence of such a mechanism. Development methods while preserving the essence of the program will be studied. At the fourth section, results will be examined. At the fifth section, conclusion and discussion be held.

2. Materials and Methods

Among the great variety of metaheuristic methods, the relatively new CSO algorithm is used in Wide range from diagnosing brain tumor [9] till classification of water quality [10]. But as shown in Deb et al.'s research [2], which is comprehensive and diligent source about CSO, it is commonly used with problems which are based upon continuous fitness functions like economic load dispatch [11], feature selection [4], reduction of energy consumption of wireless sensor network [12], improvement of concatenated convolution turbo code [13], Trajectory optimization

of hypersonic vehicles [14], localization of wireless sensor network [15] and control of fast steering mirror [16] and outperforms a number of well-known meta-heuristics in a wide range of benchmark problems [2].

It is natural that researchers do not want to limit algorithms that give successful results in wide areas such as CSO only to their own continuous areas and try to use them in different types of problems like Knapsack and TSP.

2.1.Adopting Continuous Method to Discontinuous Problem

At this juncture, adapting continuous outcomes of an algorithm like CSO into a discrete value or a job sequence can be basically divided into two different logical approaches. The first can be to preserve the original state of the continuous function and transform its outputs into a job sequence or required sequence within a certain logic. An example of this approach is Bean's Random Keys Method [6] or Huang and friends' the Smallest Position Value (SPV) method [5], which basically simplifies the Random Keys Method. SPV is a simple and quick to understand method and is based on a simple sequencing process. A basic numeric example from Huang and friends' paper [5] is given in Table 1.

Table 1. Example of the smallest position value (SPV) approach [5]

Dimension *d*	**Position Value** *(Ascending Order)*	**Job Permutation**
1	**1,35** *(5)*	**2**
2	**-2,46** *(1)*	**6**
3	**-1,52** *(3)*	**3**
4	**2,31** *(6)*	**5**
5	**0,52** *(4)*	**1**
6	**-1,68** *(2)*	**4**

In Table 1; the value of the continuous output is sorted in ascending order and the corresponding dimension d is determined as the new index value.

Another approach is to interfere with the operation of the algorithm instead of continuously generating some values and converting them to useful values. Thus, a new job sequence or a permutation will be created directly, just like Liu and friends' offered [8]. The difference between the two approaches can be seen as at what stage of the process the transformation will play the role. These two approaches are equivalent to each other, and it would be misleading to look for a significant difference between them in terms of output. As a matter of fact, the disagreement between our argument and other studies emerges at the stage after the continuous-discontinuous stage: the stage of acceptance of the new solution.

2.2. Acceptance of the New Solution

Operating fundamental of CSO lies on the movement of chickens throughout iterations and continuously scanning the area from local to global (from chick to roaster). A greedy approach as "reject this position and return to your old position" when a chicken reaches a poor fitness level compared to previous iterations is beside the point. In other words, there is no acceptance/rejection decision. In each iteration, the dice are thrown and a hierarchical structure is re-determined in the *G*'th step according to the results.

In essence, imitating nature also requires this. If an algorithm that mimic nature is sterilised, one must think twice to call the algorithm as swarm intelligence based. It is not rational to talk about a CSO which has been detached from Chicken or Swarm. Looking through Liu and Friends' [8] paper for a specific example will be appropriate. The paper is focused on solving a TSP problem with the CSO algorithm like this. While doing this, they use Swap (Swaps two selected values in a solution's sort), Order Crossover (crosses the rankings of two solutions in a given range with each other) and Reverse Order Mutation (Reverses the order between two selected points in a solution) for valid permutation in each iteration. This transformation is created based on CSO functions.

However, the authors state in a single sentence that they will make an acceptance-rejection decision by comparing the new result obtained in each iteration with the old one, but do not make any other explanations. A natural question spontaneously appeared: "why?". But there wasn't an answer in the study.

The same uncertainty is existed in other similar studies. They claimed to have applied the CSO to their problems and had good results, but they didn't mention why they installed a Greedy accept-reject mechanism that the CSO didn't have. Likewise, Mohamed [3] applied a similar method to his own problem to transform CSO's continuous algorithm into an integer vector. Although the study claims to do a Greedy search beneath the CSO, just as [2], in fact the CSO is just a small gear which is working in a Greedy machine, if you focus the big picture. Greedy's strong arm grasps the solution through the darkness, CSO only makes small retouches. It is important that better results were found compared to greedy, of course. But the issue that forms the core of this is a good understanding of whether this method is a modified CSO or a Greedy which is colored with CSO.

Han and Liu [7] also aimed to adapt the CSO algorithm to a 0-1 Knapsack problem in his study. They follow the same path as other studies, with his words "greedy algorithm is adapted in CSO to improve the quality of the feasibility". Distinctively, in the same study, Han and Liu say that the solution easily loses the diversity of the population and gets stuck in the local optimum, and therefore, it is necessary to make changes in the process. Therefore, they selectively mutate 20% of the worst fitness values.

3. Results and Discussion

In this study, a TSP model used which consist of 34 cities. The best solution to this benchmark problem is 1286. All phases of the problem coded with MATLAB R2021b. The initial solution is randomly created similar to [8] and mutated these sequences at each iteration. In order to keep the process simple and fast, only the Order Crossover Method is used among the 3 crossing over methods (Swap, Reverse Order Mutation, Order Crossover).

The process works like this: different solutions selected based on the function in the CSO algorithm and crossover the nodes determined by the same function, starting from a random point. Then the program checks order of both solutions and detects duplicated nodes and replaces with missing nodes. When the solution is found to be error-free, the fitness values are recalculated, and the hierarchical structure is renewed if the *G* value is reached. The computing times of our study are not given purposeful because the process of checking the nodes one by one and eliminating the errors is complex and multifaceted process.

First, CSO algorithm applied to problem in bare way and the outputs are obtained. A randomly generated initial solution is recalculated at each iteration based on the CSO functions, processed

without any accept-reject mechanism and A hierarchical order is updated again in each *G*'th iteration. The results of this bare transformation revealed why previous studies insisted on integrating the greedy algorithm. Results oscillated consistently between 2800 and 3800 and failed to tend any significant (positive or negative) trend. Metaphorically, the chickens wandered around blindly and could not realize the presence of food/reward. The parameters have been adjusted to stay true to the original work [1]. The average fitness of roosters (best solution found) in Swarm with 1000 iterations is presented in Figure 1.

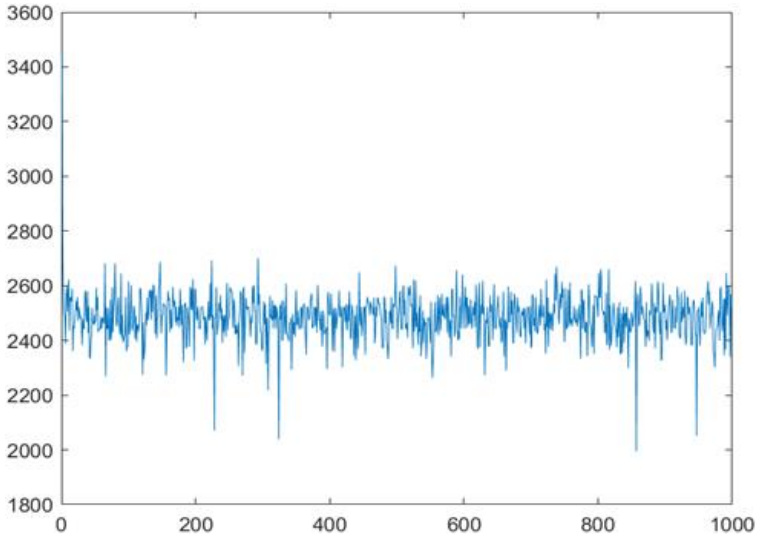

Figure 1. Bare CSO Adaptation Results (*I*=1000, *i*=30, *k*=60, *G*=20, *H*=0,8)

After all the trials, it turned out that there were 3 options. Accepting that the CSO's characteristic is incompatible with the TSP, Coupling the CSO with different methods (such as greedy search) with ignoring the fact that it is not based on swarm intelligence anymore, discovering an update in a way that doesn't break the nature of the method. At this point, idea of reconsidering the crossing over process that used during the creation of new solutions is appeared. Exchanging genes randomly with each other does not help the solution to be tended toward optimum. Instead of that, process forces the solution to oscillate at a specific range. In this case, the cross-over process updated to be unidirectional instead of bidirectional. The bidirectional crossover process shown in Figure 2.

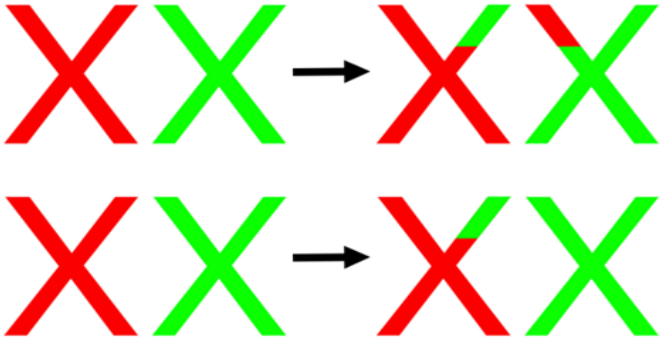

Figure 2. – Difference Between Bidirectional and Unidirectional Crossing Over

As shown on Figure 2, the weak chickens (bad fitness value solutions, represented by red chromosome) will randomly inherit the genes of the strong ones (represented by green chromosome) but they won't give genes to the strong ones. When this approach was applied, the results were surprising. All sub-groups in the swarm quickly began to tend towards the optimum

value. Some of these have done quite well. Figure 3 represents the average value of all sub-swarms (*k*=60).

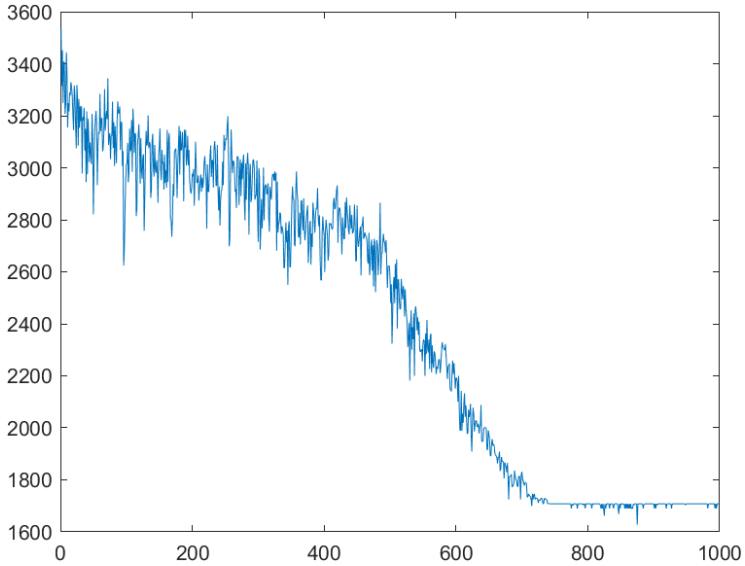

Figure 3. Result of Unidirectional Crossover Process (*I*=1000, *i*=30, *k*=60, *G*=20, P=500, I=4000, *H*=0,8)

In this figure 3, it is clearly shown that results converging directly to optimum value. But as mentioned in the previous chapters, we had a similar problem with other studies. In all trials, the program got stuck at local optima. Sometimes it was relatively close to the optimum, sometimes far from it but anyway it lost its efficiency after a certain iteration. To solve this problem, also hired the approach which introduced at [7] and randomly reordered the worst *T* number of solutions at certain *P* periods. Value of *T* is determined randomly in each period. This tweak contributed somewhat to the improvement of the solution. In Figure 4, the solution graph which obtained with the parameters *P=500, I=4000* is presented. It also represents the average value of all sub-swarms (*k*=60).

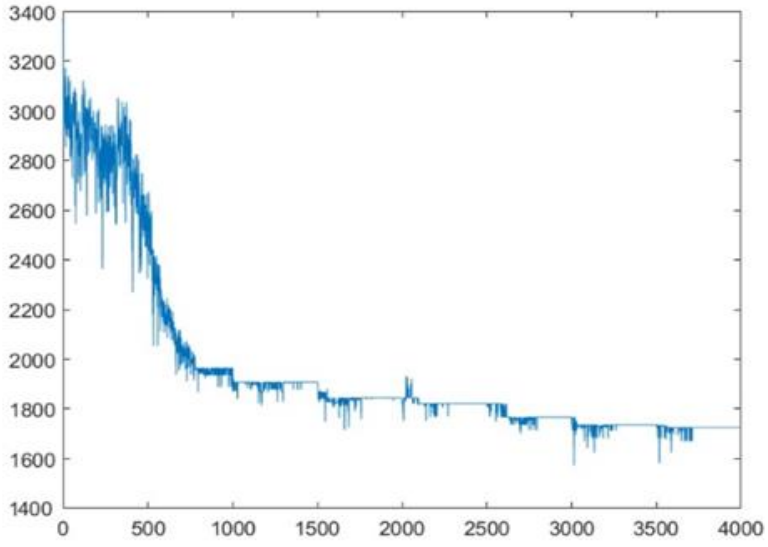

Figure 4. Result of Random Recombined Solutions per P time (*I*=1000, *i*=30, *k*=60, *G*=20, P=500, I=4000, *H*=0,8)

The wriggling in the graph every 500 iterations can be shown clearly on the Figure 4. Because a certain number of solutions are randomly recombined, the mean solution may be deteriorated for a

while. The amount of this deterioration determined by how many solutions are recombined. Since this number is determined randomly each time, some changes may be bigger than others.

4. Conclusions

Although the cleverly designed CSO algorithm delivers stunningly fast and accurate results for most types of problems, due to its nature, it had difficulty in finding an answer to a ranking problem such as TSP with its plain state. Despite the all updates, plugins and parameter arrangements (without touching the core), the results it produces are far from satisfactory.

Moreover, the difficulty in coding all these adaptations brings a serious cost. For example, our code with all the plugins to create the graphic in Figure 3 has exceeded 500 lines, but keep in mind that this information is subjective and can be improved.

When the program ran with *I=100000* to see the direction of movement, the calculation took over 1 hour and showed a slight slope towards the optimum value. However, the best solution (1396) it could find was weak compared to other methods.

On the bottom line, the main finding of the study is that the CSO method is not suitable for a recombining problem such as TSP. Although good results have been found in previous studies, reader must be aware of that the leading role in these results is not the CSO, but the second method which is reported as a footnote (eg Greedy Algorithm). This may sound like a trivial and ignorable misstatement but ranking this information into scientific indexes can be misleading for the literature and researchers.

5. References

[1] Meng, Xianbing, et al. *A new bio-inspired algorithm: chicken swarm optimization*. In: International conference in swarm intelligence. Springer, Cham, 2014. p. 86-94.
[2] Deb, Sanchari, et al. *Recent studies on chicken swarm optimization algorithm: a review (2014–2018)*. Artificial Intelligence Review, 2020, 53.3: 1737-1765.
[3] Mohamed, Taha M. *Enhancing the performance of the greedy algorithm using chicken swarm optimization: An application to exam scheduling problem*. Egyptian Computer Science Journal, 2018, 42.1: 1-17.
[4] Hafez, Ahmed Ibrahem, et al. *An innovative approach for feature selection based on chicken swarm optimization*. In: 2015 7th international conference of soft computing and pattern recognition (SoCPaR). IEEE, 2015. p. 19-24.
[5] Huang, Ko-Wei, et al. *A hybrid crow search algorithm for solving permutation flow shop scheduling problems*. Applied Sciences, 2019, 9.7: 1353.
[6] Bean, James C. *Genetic algorithms and random keys for sequencing and optimization*. ORSA journal on computing, 1994, 6.2: 154-160.
[7] Han, Meng; liu, Sanyang. *An improved binary chicken swarm optimization algorithm for solving 0-1 knapsack problem*. In: 2017 13th International Conference on Computational Intelligence and Security (CIS). IEEE, 2017. p. 207-210.
[8] Liu, Yuanjie; Liu, Qiang; tang, Zhi. *A discrete chicken swarm optimization for traveling salesman problem*. In: Journal of Physics: Conference Series. IOP Publishing, 2021. p. 012034.

[9] Taie, Shereen A.; Ghonaim, Wafaa. *CSO-based algorithm with support vector machine for brain tumor's disease diagnosis.* In: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE, 2017. p. 183-187.
[10] Sutoyo, Edi, et al. *Application of adaptive neuro-fuzzy inference system and chicken swarm optimization for classifying river water quality*. In: 2017 5th International Conference on Electrical, Electronics and Information Engineering (ICEEIE). IEEE, 2017. p. 118-122.
[11] Muralikrishnan, N. *Chicken swarm optimization for economic dispatch with disjoint prohibited zones considering network losses.* Journal of Applied Science and Engineering Methodologies, 2016, 2.2: 255-259.
[12] Wang, Qingxi; zhu, Lihua. *Optimization of wireless sensor networks based on chicken swarm optimization algorithm*. In: AIP conference proceedings. AIP Publishing LLC, 2017. p. 020197.
[13] Banerjee, Subhabrata; chattopadhyay, Sudipta. *Improved serially concatenated convolution turbo code (SCCTC) using chicken swarm optimization*. In: 2015 IEEE Power, Communication and Information Technology Conference (PCITC). IEEE, 2015. p. 268-273.
[14] Li, Yongtao; wu, Yu; qu, Xiangju. *Chicken swarm–based method for ascent trajectory optimization of hypersonic vehicles*. Journal of Aerospace Engineering, 2017, 30.5: 04017043.
[15] Al Shayokh, Md; shin, Soo Young. *Bio inspired distributed WSN localization based on chicken swarm optimization*. Wireless Personal Communications, 2017, 97.4: 5691-5706.
[16] Ren, Wei, et al. *Identification of fast-steering mirror based on chicken swarm optimization algorithm.* In: IOP conference series: earth and environmental science. IOP Publishing, 2017. p. 012086.