Parallel implementation of Metaheuristics for Optimizing Tool Path Computation on CNC Machining

Hector Rico-Garcia[a], Jose-Luis Sanchez-Romero[a,*], Hector Migallon-Gomis[b], Ravipudi Venkata Rao[c]

[a]Department of Computer Technology, University of Alicante, 03690 San Vicente, Spain

[b]Department of Computer Engineering, Miguel Hernandez University, 03202 Elche, Spain

[c]Sardar Vallabhbhai National Institute of Technology, Surat 395 007, Gujarat State, India

[*]Corresponding author
hector.rico@gmail.com; sanchez@dtic.ua.es; hmigallon@umh.es; ravipudirao@gmail.com

Abstract. The incorporation of technological advances in industry is a must, even for traditional sectors where most companies are SMEs and investments are limited. Technology can be used to increase productivity and the quality of the manufactured product. Drilling is a common procedure in industry. It usually consists of multiple drilling of a flat surface with a tool. Usually the tool is placed on the surface to be drilled at a safe distance and then it makes the drilling in a linear fashion. Optimization of the tool path often involves reducing the movement of the tool to place it over the next point to be drilled, known as airtime. The problem of minimizing airtime for drill paths is highly complex. Most proposals to solve the problem try to adapt it to the formulation of the Traveling Salesman Problem (TSP), in which the objective is to navigate a list of nodes using the minimum global distance. In this paper, the purpose is to provide a solution to the TSP applied to tool path optimization by means of a Discrete version of the Teacher-Learner-Based Optimization (TLBO) algorithm. To improve performance, the algorithm is implemented using a parallel Computer Unified Device Architecture (CUDA) and run on a manycore Graphical Processing Unit (GPU). The results show that the parallel implementation of Discrete TLBO is faster than 9x the sequential implementation.

***Keywords*:** CAM, Tool path computation, Travelling Salesman Problem, Optimization, CUDA, Parallelism, GPU

1 Introduction

Most manufacturing industries are involved in incorporating technological advances to their processes. Traditional industrial sectors are not an exception, although its investment in technology is more limited due to economic considerations and a business point of view often based on experience and know-how of workers. One of the aims when improving these processes is devoted to reduce the production time. Optimization of machining processes plays a key role in order to meet the requirements for high precision and productivity. The productivity of machine tools can be greatly improved using CAD/CAM systems so as to generate CNC programs. These systems provide appropriate numerical control codes for different machining processes, one of which is the drilling operation.

Drilling through numerical control tools is a common procedure in most industrial processes. It usually consists of multiple drilling of a flat surface using a cylindrical or spherical tool. Although other variants may exist, the usual process using numerical control consists of placing the tool on the surface to be drilled at a safe distance and then carrying out the drilling process with a linear movement (usually with G01 movements from the G-Code or RS-274 standard) of the tool perpendicular to the surface, and then removing the tool in the same way. The drilling is carried out accurately and at the appropriate speed, always depending on both the material to be drilled and the tool, while the movement to the next drilling point is carried out at a higher speed since it is done at a distance which allows no collision with the material to be drilled (usually with G00 movements).

Since the drilling process is dependent on the material and the tool and must be adjusted to the characteristics of the industrial process itself, the optimization of the paths for this type of machining usually involves reducing what is known as “airtime”, i.e. optimizing the movement of the tool to place it above the next point to be drilled [1].

The problem of achieving a minimum airtime for drilling paths is highly complex and has been defined as NP-Complete [2]. The generation of these optimized trajectories remains an open problem for researchers.

Because of similarity, most solutions adopt strategies based on different variants of the Travelling Salesman Problem (TSP). A recent review can be found in [3], where it is found that 79% of the approaches use ad hoc algorithms to solve the TSP in different variants. Another interesting conclusion from this review indicates that most works usually test few examples, which are moreover limited by the number of holes to be performed. Unlike the TSP associated with interurban navigation, in the scope of machining there is no common database so that researchers can compare different drilling schemes to test the robustness of the algorithms developed.

The increase in the processing power of computers together with the appearance of affordable architectures equipped with multi-core processors has allowed the use of heuristic and meta-heuristic schemes to approximate the solution to this problem. For example, in [4] Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) are used due to their efficiency and flexibility to generate optimal solutions when applied to different engineering problems.

It is worthwhile emphasizing that the problem of drilling has many variants. The TSP has also been used to solve the problem of generating paths on irregular free surfaces (freeform surfaces) where the concept of curvature appears as a new complexity factor in path calculation [5].

In this work, an approach to TSP applied to tool path optimization is proposed by means of a discrete version of the metaheuristic optimization algorithm TLBO (Teacher Learner Based Optimization). Moreover, with the aim of improving performance, the algorithm has been implemented on a parallel architecture, specifically a CUDA-based implementation which runs on a GPU platform.

This paper is structured as follows: Section 2 provides a formulation of the TSP and its analogy with the problem of optimizing tool path computation and a review of the main works found in the literature about the application of different metaheuristics regarding TSP and tool path computation. Section 3 provides a description of the TLBO method in both continuous (original) and discrete versions. Section 4 explains the implementation of TLBO on a parallel CUDA architecture in order to improve performance with regard to a traditional, sequential implementation. Section 5 shows

the results of applying the parallel discrete TLBO on set of scenarios. Finally, Section 6 summarizes the findings of the research work and proposes future lines of research.

2 TSP and Metaheuristics for Tool Path Computation

The TSP belongs to the class of combinatorial problem and has been widely studied. Its definition is rather simple but obtaining an optimal solution can be very difficult depending mostly on the problem size. The problem can be formulated by means of graph theory: it is defined as a graph G = (V, A), where V = $\{v_1,...,v_n\}$ is a set of n vertices (nodes) and A = $\{(v_i, v_j)/\ v_i, v_j \in V, i \neq j\}$ is a set of edges that has an associated non-negative cost (distance) matrix D = (d_{ij}). The problem is said to be symmetric if $d_{ij} = d_{ji}$ for any pair of nodes $(v_i, v_j) \in A$, and it is said to be asymmetric if $d_{ij} \neq d_{ji}$ for some pair of nodes $(v_i, v_j) \in A$.
If $R:\{1, ..., n\} \rightarrow \{1, \ldots, n\}$ is a bijective function that defines a reordering of vertices v_i in V, the TSP is defined as minimizing the following function:

$$f_{TSP} = \sum_{j=1}^{n} d_{R(j)R(j+1)} \quad (1)$$

A correspondence between the tool path optimization in machining and the TSP can be directly devised. The set of nodes are related to the different points to be drilled and the aim is minimizing the airtime so as to increase productivity. This objective is also related to reduce energy consumption and tool wear.

Given a set of n nodes to be visited, finding the best solution by exhaustive search would involve a complexity of the order of $(n - 1)!$ That is, in case of 5 nodes, it would be necessary comparing only 4! = 24 possible solutions, but in case of 20 nodes it would imply comparing 19! = 121,645,100,408,832,000 possible solutions so as to obtain the optimal one.

A large amount of research works related to TSP has provided different methods so as to reduce this complexity. Due to the features of TSP, metaheuristic methods are suitable for dealing with the problem, which is supported by the wide variety of

research works related to TSP together with metaheuristics. Indeed, while exact resolution methods come with a proof that the optimal solution will be found in a finite, though often prohibitive, time, metaheuristic methods are suited to finding a good enough solution in a small enough computation time. Consequently, they are not subject to combinatorial explosion, that is, the situation where the computation time needed to find the optimal solution dramatically increases as an exponential function of the problem size.

In [6], a survey of swarm intelligence applied to graph search problems is provided. The analysis performed is mainly focused on Ant Colony Optimization (ACO) and Bee Colony Optimization (BCO). In [7], two variations of the Artificial Bee Colony (ABC) method are proposed for solving the TSP; experiments are carried out on a set of 15 TSP test problems; the performance of both ABC algorithms is compared with that of eight different versions of GA, and also with that of Ant Colony System (ACS) and BCO; results show the suitability of both version of ABC when applied to TSP problems. Most research devoted to provide a solution to TSP by using metaheuristic optimization methods apply the ACO method [8 – 11], sometimes in a hybrid version with other techniques [12]. Since the method mimics the behavior of ants as they search for the shortest path between the colony and a food source, it can naturally adapt to TSP where the goal is to obtain a minimum distance path.

With regard to the research on metaheuristic methods applied to the optimization of the machining process, a wide variety of works can be highlighted. In [13], it is concluded that TSP is included in 92% of the works on soft computing applied to tool path optimization. GA and hybrid algorithms are the most frequently used, with about 20% papers from the ones analyzed. About 18% of the research works uses modified algorithms to optimize tool path for drilling. Besides, about 13% of researchers use ACO to find the shortest path based on previous works. In [14], it is pointed out that 60% of research work on TSP and metaheuristics applied to tool path optimization involves the use of ACO, PSO or GA. In [15], a review of research works on tool path optimization in CNC machines is provided. An overview of different optimization methods is shown, such as Artificial Neural Networks (ANN), Artificial Immune

Systems (AIS), GA, ACO and PSO. Also, a collection of previous research on tool path computation with different types of AI optimization methods is presented to show their capability in optimizing the machining processes. Conclusions of the study remark that GA and PSO are most widely used to optimize machining efficiency. Results also allow concluding that GA has been successfully applied for many optimization problems related to tool path. The ACO method is mainly used in minimizing machining time, airtime, and reduction of tool travel path. On the other hand, ANN are used to increase productivity, increase surface quality and decrease costs. PSO is mainly employed for reducing tool travel path and cost, and minimizing machining time. In [16], research is focused on the optimization of the traversing paths of tufting tours for carpet manufacturing. An analogy of the TSP, called traversing tuft problem (TTP), is formally presented. GA is hybridized with an ad-hoc heuristic method so as to optimize paths. In [17], the general formulation of the TSP model is extended by considering the precedence of the tool operations. The objective of the model is minimizing the idle time of tools with regard to internal operations. A recent optimization algorithm, called Satin Bowerbird Optimizer (SBO), is used to cope with the problem. Since SBO is originally applied to global optimization problems, the method is modified with discretization and local search procedures. The performance of the proposed algorithm is tested on well-known precedence-constrained TSP benchmark problems by comparing it with other meta-heuristic approaches: Adaptive Evolutionary Algorithm (AEA) and ACS. Results show that the proposed method outperforms the other algorithms. When applied to real life hole drilling scenarios, up to 4.05% improvement in operational time is achieved. In [18], a mathematical model of the path optimization problem is proposed, as well as the use of ACO based on clustering analysis for minimizing auxiliary time of the path optimization. When massive holes are machined, they are divided into many sub-regions using an improved k-means clustering method. Machining efficiency is increased by 18.51% with regard to the original path and improved by 2.08% with regard to the traditional ACO algorithm. However, only a TSP problem is used for testing. In [19], a method is presented which uses heuristic optimization techniques to solve TSP for segments. The proposed method adopts PSO and GA, which are solely implemented in MATLAB. The results of the proposal are compared with those of

two industry standard CAM systems. Using the proposed optimization method saves up to 40 % of the tool airtime during machining. In [5], an approach for compound surface finishing by treating the tool path planning task as a TSP is presented. A compound surface is usually composed of several patches of trimmed or untrimmed surfaces. The concept of curvature map is proposed. With this concept, the curvatures of the surface patches are associated with the corresponding cells of the map, where the path intervals are determined at a later stage. The Cutter Contact points and the normal vectors are calculated on the mesh model of the compound surface with a linear algorithm. The obtained Cutter Contact points are linked as nodes in LKH, in which the distance function is redefined to cope with the illegal linking problems. With LKH, tool retractions are no longer necessary. The resulting tool path is capable of covering the whole compound surface in only one pass. This method is very suitable for irregular-shaped compound surfaces, especially when holes appear. In [20**]**, GA are used so as to generate sub-optimal tool paths. The method is tested with three examples from TSPLIB (QA194, XIT108, and FI10639). The results show a fitness of 96%, 95% and 96% respectively. In [21**],** a path-planning optimization study for CNC machining center devoted to machining jobs involving a large number of holes to drill, mostly arranged in concentric circular patterns, is proposed. The paper proposes a hybrid ACO developed to take advantage of the geometric hole-pattern arrangement, as well as local search. Results of simulation show that the proposed approach achieves higher performance compared to the classic ACO approach, to GA, and to the simple spiral path generated by means of commercial CAD software. The approach is then applied to the drilling path planning of a 1,000 holes food-industry separator plate. Numerical simulations of example studies showed a higher performance of the modified ACO compared with GA and the basic ACO, with up to 15% reduction in total travel distance from the default spiral path in complex hole layouts. A model of a full-scale food separator screen with 2,100 holes was used to demonstrate the scalability of the proposed approach. In [22], an application for tool path optimization of multi position hole machining using GA is developed, which is capable to deal with G-code files as input data for any component to be machined on CNC machining centers. The results of GA applied to the case of a series of nodes are compared with those determined by the manual calculation of all

possible paths, and therefore the minimum of them. Both methods gave exactly the same results which demonstrate the robustness of the GA algorithm. Comparing the results of GA and branch-and-bound methods for different cases, GA provides better results than branch-and-bound in the case of irregular position patterns and equal results in the cases of regular patterns. The results of a 50-position case study conducted by 25 CNC programmers show that they have only a 4.26% chance of achieving the optimal solution. In [23**],** GA with modified crossover method is used so as to cope with a drilling problem involving an energy efficient path. Results show that the energy efficient tool path consumed less energy than the tool path optimized for minimum time. In [24], a hybrid GA is proposed to optimize tool path; the initial seed solution is generated by special heuristic and combined with random initial solution generated by simple GA. An index known as Relative percentage deviation (RPD) is defined and used for analyzing the results by varying the size of the jobs. The tool gets retracted and repositioned several times in multi pocket tasks during rough machining. Depending on the complexity of the job, these actions consume from 15% to 30% of total machining time. Three types of problems have been considered for comparing hybrid GA and simple GA: the number of nodes determines whether the problem is hard, medium or easy (275, 150 and 85 nodes respectively). Results show that hybrid GA performs well for all three easy, medium and hard sized problems. As the number of nodes increases, the hybrid GA proves to be more effective in minimizing non-productive machining time, taking into account the calculation time limit as a stop criterion. In [25], simulated annealing (SA), GA and a hybrid algorithm (hybrid-GASA) combining both techniques are applied to tool-path optimization problems for minimizing airtime during machining. These three algorithms were tested on three-axis-cartesian robot for wood materials milling. Their performances were compared with a minimum path and, therefore, minimum airtime. Experimentation on two examples shows that hybrid-GASA provides better results than the other heuristic algorithms working alone. The hybrid approach produces about 1.5% better minimum path solutions than simple GA and 47% better minimum path solutions than simple SA. With regard to a random selection of the tool path, in the hard case reduction of machining time reaches up to 84.1%, and in the easy case the reduction is 79.48%. In [26], a methodology to generate optimal or sub-optimal

sequences of G-commands to minimize the manufacturing time is proposed. The solution starts from original G-codes provided by a CAD/CAM software tool. For optimizing the time along the travel path, an implementation of ACO known as Parallel ACO (P-ACO) is developed which allows achieving the optimization task efficiently by speeding up the original ACO. Processors used vary from 1 up to 6. Results are verified using a professional CNC machine. The analysis of numerical results demonstrates that the use of the proposed methodology provides improvements of up to 62% over solutions obtained by CAD/CAM commercial software.

As a summary of this review, it can be concluded that metaheuristic methods applied to the optimization of the tools' trajectory is an area of productive research and work continues in this area. However, it is not easy to find work that includes the parallel implementation of these techniques with regard to their application to machining processes. This situation leads to the idea that it is necessary to investigate the parallelization of these processes, given that there are currently parallel hardware platforms of affordable cost with a large computing power, whose characteristics in many cases make them very suitable for addressing problems of metaheuristic optimization.

3 The TLBO optimization method

3.1 Original TLBO Definition

TLBO is a metaheuristic optimization method based on the teaching-learning process which involves a *teacher* and a set of students (*learners*), in such a way that knowledge is increased by means of an iterative series of stages where each *learner* learns from the *teacher*, but also from another *learners* [27]. Instead of the natural goal of increasing knowledge, the aim of TLBO is the optimization of a function *f*. A population of individuals is initially created, and each one of them is then assigned a random value for each one of the design variables defined in function *f*.

After the initial assignment of parameters to learners, the method iterates through the following two stages until a stop criterion is fulfilled:

3.1.1 Teacher stage. In this stage, each individual *i* evaluates the function *f* with its own parameters, *f* **(***X*(*i*)), and the one with the optimal (usually minimal) evaluation is designed as the population *Teacher*. Each individual updates its parameters according to the ones of the *Teacher* and the mean population parameters. The *Teacher* (X_{best}) of the current generation is therefore used to create a new version of each individual (X_{new}) according to the following equation:

$$X_{new}(i,j) = X(i,j) + rand(0,1)(X_{best}(j) - TFactor \cdot X_{mean}(j)) \quad (2)$$

In (1), *X*(*i, j*) is the design variable *j* of individual *i*. It is modified by using the value of variable *j* from the *Teacher*, $X_{best}(j)$, the variable mean from the whole population, $X_{mean}(j)$, and the *TFactor*. The *TFactor* parameter adopts the integer value 1 or 2 according to the following expression:

$$TFactor = round(1 + rand(0,1)) \quad (3)$$

After modifying the variables of an individual *i*, it evaluates the function again. In case that the evaluation of *f* ($X_{new}(i)$) provides a better (lower) result than that of the original individual, the new values of parameters replace the old ones for that individual *i*.

3.1.2 Learner stage. After the teacher stage, each individual is compared with a random contestant from the population. The individual with a better function evaluation is designed as the *Best Learner*, being the other designed as the *Worst Learner*. Their parameters are used to generate a new individual by means of the following expression:

$$X_{new}(i,j) = X(i,j) + rand(0,1) \cdot (BestLearner(j) - WorstLearner(j)) \quad (4)$$

When every variable *Xnew*(*i, j*) is generated, this new individual evaluates the function. If this evaluation is better than that of the original one, the values of the old one are replaced by the new values in the design variables.

The main advantage of TLBO over other metaheuristic algorithms is that has no algorithm-specific control or tuning parameters. Indeed, only population size and generations, e.g., number of iterations, should be configured. In recent research works, it has been demonstrated that TLBO is more efficient than other optimization methods [28 – 31]. Hence, research related to TLBO has reached a remarkable development in the last years, and it keeps on be studied, improved and applied in a wide range of scientific and engineering scopes.

3.2 Discrete TLBO

The original version of TLBO was developed for being applied to functions or problems where design variables are in the continuous domain. On the other hand, combinatorial problems are those which involve finding a grouping, ordering, or assignment of a discrete, finite set of objects that satisfies a set of conditions. TSP is a combinatorial problem since its solution is a reordering of the whole set of vertices from a graph. When applying TLBO to TSP, an in-depth modification must be made so as to adapt the continuous formulation to discrete, combinatorial problems. Some research works can be found that proposes and approach to TLBO for discrete problems, being the resulting method called *Discrete-TLBO* (DTLBO).

In the current work, an implementation of the DTLBO based on the work developed in [32] has been carried out, although several relevant modifications have been performed.

3.2.1 Representation of individuals. Each individual is a sequence of nodes, representing a solution to be evaluated. As an example, if the problem deals with 8 nodes (v_0, v_1... v_7) to be navigated, an individual could be represented as shown in Fig. 1. The solution represented in Fig. 1 consists of starting in node v_6 and finishing in node v_0 before returning to the starting node v_6, visiting thus the different nodes following the order $v_6 \rightarrow v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_7 \rightarrow v_3 \rightarrow v_4 \rightarrow v_0$.

In [32], the initial assignment of individuals is performed by random permutations of the nodes to be visited and the global population is divided into subpopulations in order to avoid becoming prematurely trapped in local minima, concretely four subpopulations are considered. In the current work, a modification is proposed which consists in using a greedy strategy when generating one of the individuals of each subpopulation, while the others are randomly created; in this way, a sub-optimal solution is generated as a starting point, which can accelerate convergence towards an optimal solution.

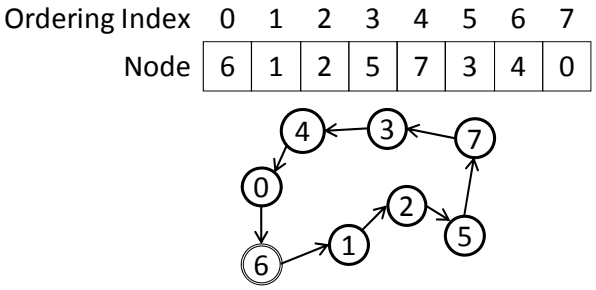

Fig. 1. Representation of an individual solution.

3.2.2 DTLBO Teacher stage. A *Partial Teacher* with the best evaluation is determined from each subpopulation. This individual will be used so as to update learners in the subpopulation. The mean values within each subpopulation are also gathered into a *Mean* individual that will also be used for updating each learner. A global *Teacher* is also determined as the individual with the best evaluation within the whole population.

It should be pointed out that, while the *Teacher* is always a valid individual, the *Mean* individual could not be, since some nodes could be repeated and others could not appear. Therefore, a feasibility operation must be carried out on this *Mean* individual so as to convert it to a valid individual. This feasibility operation is as follows:

Step 1: The nodes that did appear in the individual are directly represented in vector *TempFeasibleInd*. Those nodes which appeared more than once occupy the last component where they appeared. Otherwise, the vector component is set to empty (symbol −). In the example shown in Fig. 2, nodes 1 and 7 appear twice, while nodes 0, 2, 3, and 5 appear just once.

Step 2: Determine the nodes that did not appear in the individual. For the individual in Fig. 2, they would be nodes 4 and 6. Each node that did not appear is written into the first free component, obtaining thus the resultant feasible individual.

Crossover operation. Symbol ⊗ represents the crossover operation in (5). There are four different crossover operations that can be selected so as to create a new individual:

$$Xnew(i) = X(i) \otimes Teacher \tag{5}$$
$$Xnew(i) = X(i) \otimes PartialTeacher(i)$$
$$Xnew(i) = X(i) \otimes Mean(i)$$
$$Xnew(i) = PartialTeacher(i) \otimes Mean(i)$$

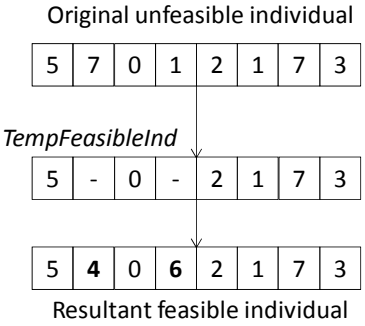

Fig. 2. An example of the feasibility operation.

In the current work, a random selection of each one of the above crossover operations is performed each iteration and for each individual. This strategy is different from the one used in [32], where each subpopulation is assigned a fixed calculation for the crossover operation. With the change proposed in this work, a wider variety of individuals is achieved and, therefore, a new strategy is added to prevent a subpopulation from being prematurely trapped in a local minimum. By introducing this random selection, new solutions appear since the crossover for each individual within a subpopulation could be different each one of the iterations.

The crossover operation works in the following way: given two individuals, X and Y, a new individual X_c is generated; a starting and an ending position in the order of visiting the nodes are randomly selected, so that individual X replaces its components by the ones of individual Y. Fig. 3 shows an example of this operation. Since the creation of

a new individual by crossover can produce an invalid individual, the feasibility operation must be performed on it.

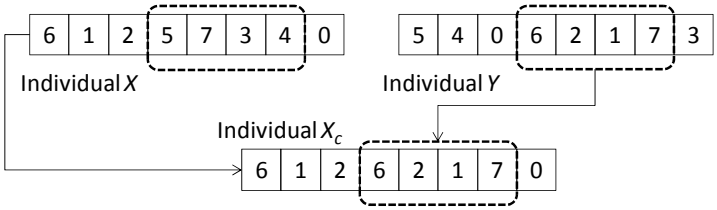

Fig. 3. An example of the crossover operation.

Mutation operation. After the new individual is created crossover, a mutation is applied on it. Symbol Θ is used so as to indicate the mutation operator in (6). The mutation is performed as follows: a starting and an ending position in the order of visiting the nodes are selected; then the elements are flipped between the two positions so as to generate a mutation.

$$X_{cm}(i) = \Theta X_c(i) \tag{6}$$

As an example, given an individual X_c, a starting position 3 and an ending position 6 in the order of visiting the nodes are randomly selected; the mutated individual X_{cm} is obtained by the mutation operation as shown in Fig. 4:

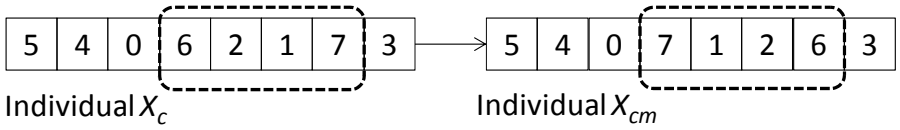

Fig. 4. An example of mutation operation.

3.2.3 DTLBO Learner stage. After the teacher stage, the learner stage is performed. Like in the original TLBO, for each individual $X(i)$ a learner $X(k)$ is randomly selected within its subpopulation. The learner $X(i)$ is updated as indicated in (7).

$$Xnew(i) = X(i) \otimes X(k) \quad (7)$$

where $\otimes$ represents the crossover operation, working in the same way as the crossover operation in the teacher stage.

Once the crossover operation is carried out, the feasibility operation is performed again on the new individual. Then the mutation operation is carried out just the same way as in the teacher stage.

In [32], DTLBO is compared to ACO, ABC, PSO, and GA when facing different TSP instances; conclusions remark that DTLBO achieves better approximate solutions than the other algorithms in most cases, being its percentual relative error from 0.01 to 1.64 in those cases. ABC achieves better results than DTLBO only in two instances and ACO achieves the same result as DTLBO in one TSP instance. Conclusions also highlight that DTLBO performance decreases when dealing with very large problems; therefore, it is suggested that efforts should be made to focus on strategies to improve DTLBO performance for those larger TSP problems.

4 Parallel TLBO implementation on GPU

The modifications made on the original TLBO so as to adapt it to combinatorial problems involve adding more complexity and computational cost to the different stages. Therefore, DTLBO iterations are much more costly than those of the original TLBO, which may worsen the overall execution time. To minimize the impact of these changes on the algorithm performance, a parallel version of DTLBO has been implemented using CUDA architecture and run on a GPU platform. It must be taking into account that parallelizing an algorithm by means of CUDA not always means a better performance. In case of DTLBO, its specific features can be exploited to achieve substantial performance improvement over sequential solutions using this parallel architecture.

4.1 Design of the memory organization

An essential step when facing the parallelization of the algorithm is to devise a correct design of the memory organization and the execution flow in order to minimize the global thread locks. If GPU memory is poorly managed, the result can be a negative impact on the execution time, since transfer operations between memory levels within the GPU are time-consuming and, therefore, they must be minimized. Therefore, the GPU memory levels will be organized so that each of them manages different type of information (thread, subpopulation, and global levels) as depicted in Fig. 5.

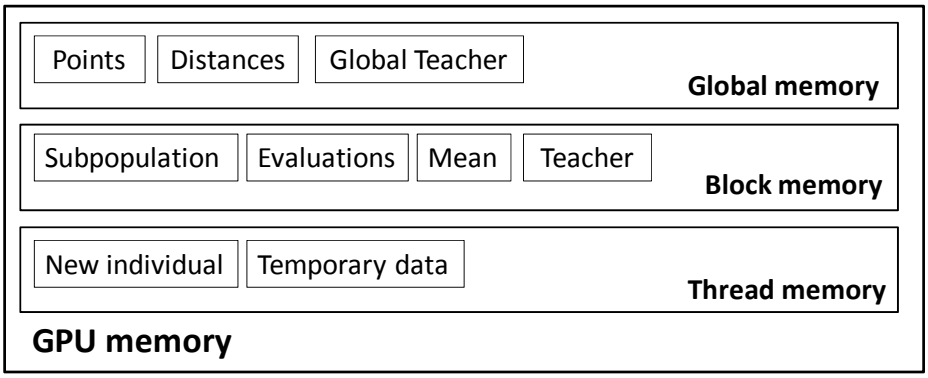

Fig. 5. Organization of CUDA memory.

Global memory. Two main blocks of information are stored in global memory: an array where all the necessary points for the generation of the individuals of the populations in each block are stored; and an array with all the pre-calculated distances between points. Global memory also stores the best individual of the whole population together with its evaluation (global *Teacher*) in each iteration; it must be global so as to be used by the whole population. Most information stored in global memory is read-only and is used to avoid distance calculations within the evaluation of each individual and to save memory in the individuals of each subpopulation; only the data about the best individual (global *Teacher*) will be modified if necessary each iteration.

Block memory. The information shared by a subpopulation is stored in block memory. This storage is organized as a matrix where each row represents an individual

and each column represents the index of a point; an additional column stores the individual's solution to avoid repeating the evaluation.

Thread memory. The thread local memory store variables which are used for the required calculations within each phase of the algorithm and updated each one of the iterations. This memory is private to each thread and is not shared with the rest of the population.

4.2 Execution flow

An execution flow has been designed in order to minimize the blockages of the threads when synchronizing during the execution of the algorithm. TLBO involves a series of stages that force the threads to be synchronized so as to obtain common information for the whole population, such as the mean individual and the best individual (*Teacher*). Moreover, the reduction technique is applied to minimize the iterations required to obtain these values from the population. In order to parallelize the execution of the algorithm, each thread is used as an individual of the population. The first thread (with index 0) of each sub-population is responsible for performing the operations in local memory, while the first thread of the first population is responsible for performing the necessary operations in global memory. This is depicted in Fig.6. In this way, conflicts and delays when accessing the different memory levels are avoided.

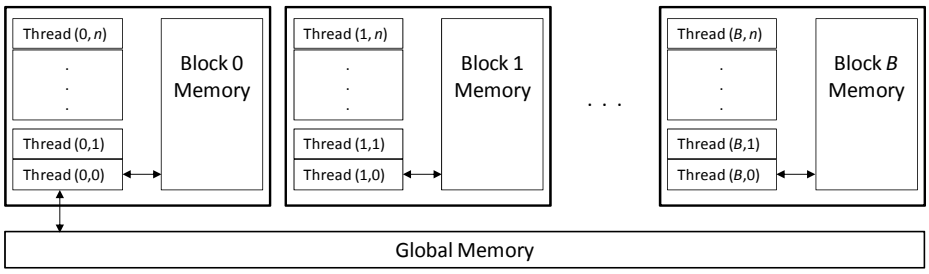

Fig. 6. Subpopulations organization and memory accesses.

5 Experimentation

Experiments are focused on comparing the computation time when solving four well-documented TSP problems from the TSPLIB library [33] and also a Printed Circuit

Board (PCB) file example. A comparison is made between a sequential implementation and a manycore GPU implementation using CUDA. The hardware used for the experimentation is a Pentium i7 processor at 3.2GHz with an NVIDIA GeForce 1060p graphics card, 6 GB GPU RAM and 32 GB DDR4 RAM. The CUDA platform version was 9.2 compiled on Visual Studio 2017 with the most recent Windows C++ runtime available. Different scenarios are defined for each problem by modifying the population sizes (64 and 100 individuals) as well as the number of iterations to run the algorithm (1,000, 5,000 and 10,000 iterations). The problems from TSPLIB which are used for experimentation are KroA100, KroB100, KroC100, with 100 nodes, and Lin105, with 105 nodes. KroA100, KroB100, and KroC100 belong to the Krolak/Felts/Nelson set. Lin105 represents the conversion to Euclidean 2D space of the positions of 105 cities of the Lin/Kernighan problem, extracted from the original problem with 318 cities. These problems have been used as a benchmark not only to measure the computation time, but also to evaluate the approach to the best solutions since these problems have documented those optimal. Experimentation has also included a "Centroid" file. This type of file defines the positions of the electrical components on a PCB design. From this file, the components that should be soldered have been extracted and a path has been created to optimize the soldering time of the components to the board. The number of nodes is 74. In this case, the optimal path is unknown.

Figures 7-11 show the results with regard to the CPU and GPU average time in milliseconds of 50 runs of DTBLO applied to every problem with the different scenarios. Results demonstrate that the GPU parallel implementation of DTLBO achieves an improvement in performance higher than 9x in case of a large number of individuals and iterations. Table 1 shows the optimal for every problem (with the exception of Centroid), the solutions achieved by parallel DTLBO, and the corresponding percentual relative errors. It can be observed that the approximation of the DTLBO solution to the optimal of the different problems is very satisfactory. The paths obtained for the different problems are shown in Figures 12 and 13. Moreover, the Centroid path obtained by DTLBO has been simulated with the open-source CAMotics tool, which emulates the execution of 3-axis G-Code programs for CNC and displays the results in 3D (see Figure 13 right).

Table 1. Optimal, DTLBO solution, and percentual relative error for the TSP tested.

	KroA100	**KroB100**	**KroC100**	**Lin105**	**Centroid**
Optimal	21,282	22,141	20,749	14,379	Unknown
DTLBO	21,294	22,169	20,782	14,391	279.01
% Relative error	0.056	0.126	0.159	0.083	Unknown

Given the characteristics of the GPU and the possibility of processing a large number of threads per block, it can be seen that the increase in population does not have as much time penalty as the CPU, being the GPU times only affected by the number of iterations. In addition, the GPU block architecture is well suited for algorithms with subpopulations, since they can be placed on different GPU blocks and, therefore, run in parallel if the number of subpopulations allows for it. For real-world applications, different problems can be run on the same GPU, maximizing GPU utilization and minimizing system response time.

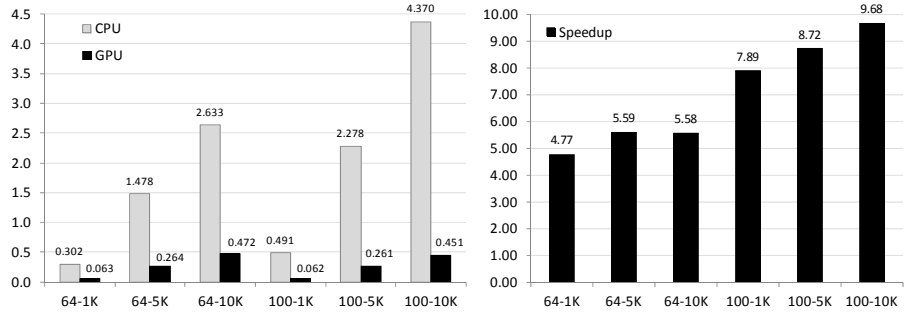

Fig 7. Results from KroA100 with regard to different number of individuals (64 and 100) and iterations (1000, 5000 and 10000). CPU and GPU time (left); corresponding speedup (right).

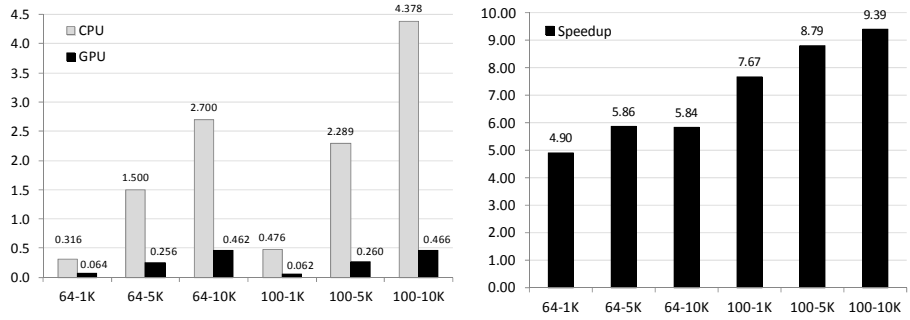

Fig 8. Results for KroB100 with regard to different number of individuals (64 and 100) and iterations (1000, 5000 and 10000). CPU and GPU time (left); corresponding speedup (right).

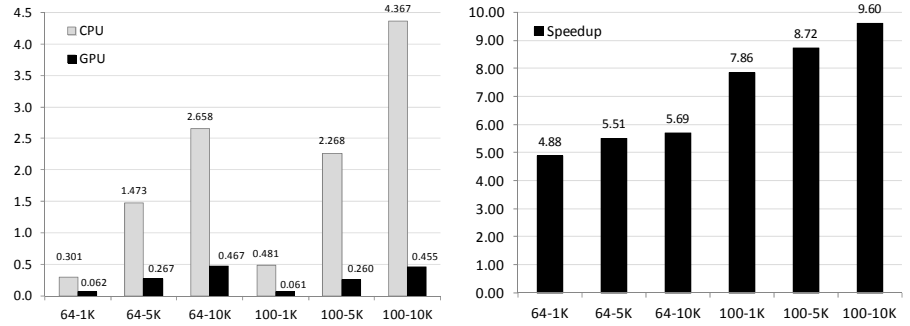

Fig 9. Results for KroC100 with regard to different number of individuals (64 and 100) and iterations (1000, 5000 and 10000). CPU and GPU time (left); corresponding speedup (right).

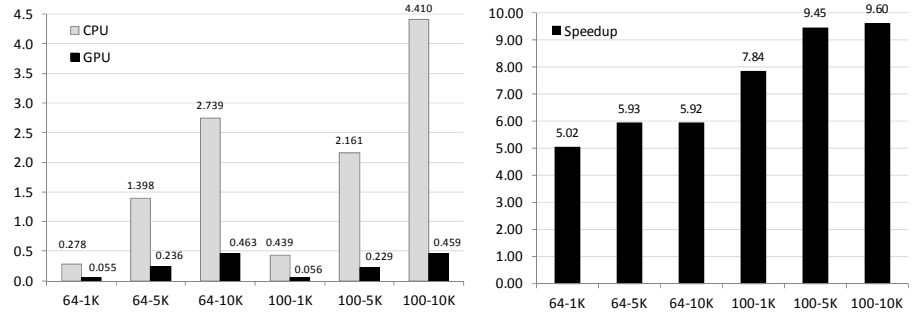

Fig 10. Results for Li105 with regard to different number of individuals (64 and 100) and iterations (1000, 5000 and 10000). CPU and GPU time (left); corresponding speedup (right).

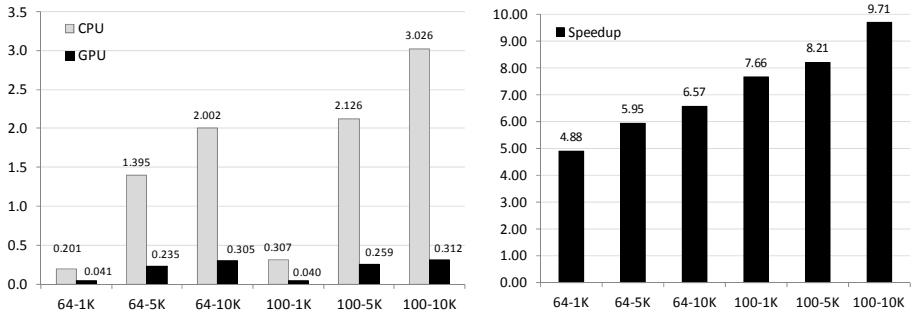

Fig 11. Results for Centroid with regard to different number of individuals (64 and 100) and iterations (1000, 5000 and 10000). CPU and GPU time (left); corresponding speedup (right).

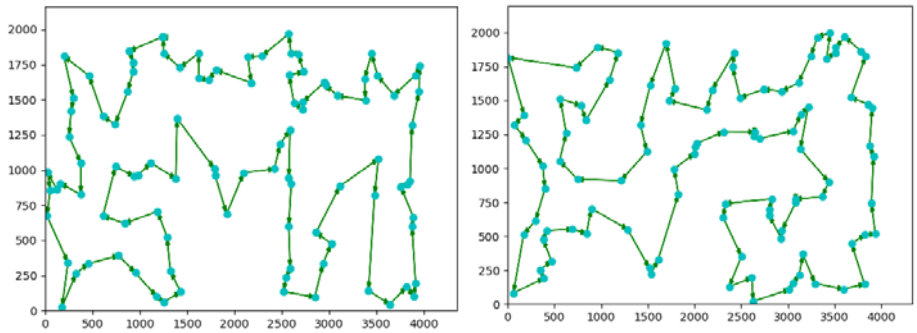

Fig. 12. DLTBO solution for KroA100 (left) and KroB100.

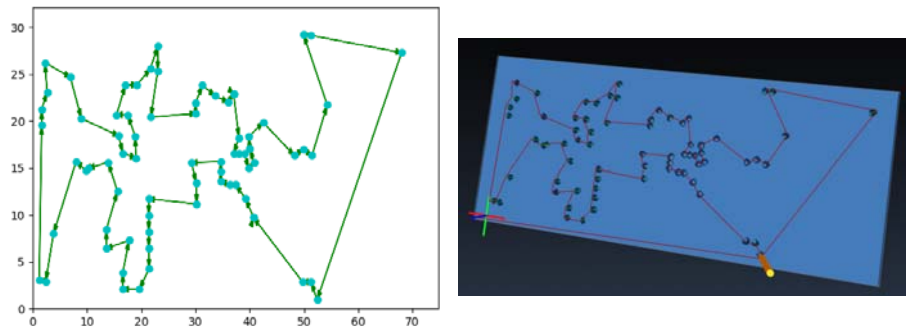

Fig. 13. DTLBO solution for Centroid (left); result of the simulation with CAMotics.

6 Conclusions

In this paper, a parallel implementation of the discrete Teaching-Learning-Based Optimization algorithm applied to the Traveling Salesman Problem using a manycore

GPU approach to improve performance is proposed. This algorithm has proven in previous works to be a good choice for solving TSP problems. Since the algorithm complexity is increased when transformed so as to cope with combinatorial problems, the parallel, manycore GPU implementation is used so as to improve substantially the speed of the algorithm, obtaining significant speedups with regard to a sequential implementation. In case of a large number of individuals and iterations, the speedup reaches more than 9x. These results make the algorithm suitable to be applied in problems where the number of points to be visited following an optimized path is very high. The implementation developed could still be improved trying to emphasize the thread blocks to optimize the use of GPU resources. Moreover, in the TSP instances tested the percentual relative error achieved with DTLBO is very satisfactory.

The parallel implementation of DTLBO on CUDA has been carried out on an affordable computer and, therefore, it can easily be applied to industrial sectors that cannot cope with large investments in technology. As a continuation of this research, it would be interesting to further improve the algorithm in its various stages and in the use of GPU resources. Also, it would be interesting to carry out tests in real machining environments to check if the results obtained are viable when they are manufactured.

References

[1] Castelino, K. D'Souza, R., Wright, P.K.: Toolpath optimization for minimizing airtime during machining. Journal of Manufacturing Systems 22, 173–180 (2003). https://doi.org/10.1016/S0278-6125(03)90018-5

[2] Lewis, H.: Michael R. ΠGarey and David S. Johnson. Computers and intractability. A guide to the theory of NP-completeness. W. H. Freeman and Company, San Francisco1979, x 338 pp. Journal of Symbolic Logic 48(2), 498-500 (1983). doi:10.2307/2273574

[3] Dewil, R., Küçükoğlu, İ., Luteyn, C., Cattrysse, D.: A Critical Review of Multi-hole Drilling Path Optimization. Archives of Computational Methods in Engineering 26 449–459 (2019). https://doi.org/10.1007/s11831-018-9251-x

[4] Karuppanan, B.R.C., Saravanan, M.: Optimized sequencing of CNC milling toolpath segments using metaheuristic algorithms. Journal of Mechanical Science and Technology 33, 791–800 (2019). https://doi.org/10.1007/s12206-019-0134-3

[5] Lin, Z., Fu, J., Sun, Y., Gao, Q., Xu, G., Wang, Z.: Non-retraction toolpath generation for irregular compound freeform surfaces with the LKH TSP solver. International Journal of Advanced Manufacturing Technology 92, 2325–2339 (2017). https://doi.org/10.1007/s00170-017-0247-8

[6] Ilie, S.V.: Survey on distributed approaches to swarm intelligence for graph search problems. Annals of the University of Craiova-Mathematics and Computer Science Series 41(2), 251-270 (2014).

[7] Karaboga, D., Gorkemli, B.: Solving traveling salesman problem by using combinatorial artificial bee colony algorithms. International Journal on Artificial Intelligence Tools 28(01), 1950004 (2019).

[8] Gan, R., et al.: Improved ant colony optimization algorithm for the traveling salesman problems. Journal of Systems Engineering and Electronics 21(2), 329- 333 (2010).

[9] Shokouhifar, M., Sabet, S.: PMACO: A pheromone mutation based ant colony optimization for traveling salesman problem. In: 2012 International Symposium on Innovations in Intelligent Systems and Applications. IEEE (2012).

[10] Bai, J., et al.: A model induced max-min ant colony optimization for asymmetric traveling salesman problem. Applied Soft Computing 13(3), 1365-1375 (2013).

[11] Głąbowski, M., et al.: Shortest path problem solving based on ant colony optimization metaheuristic. Image Processing & Communications 17(1-2), 7-17 (2012).

[12] Mahi, M., Baykan, O. K., Kodaz, H.: A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. Applied Soft Computing 30, 484- 490 (2015).

[13] Abidin, N. W. Z., Ab Rashid, M. F. F., Mohamed, N. M. Z. N.: A review of multi-holes drilling path optimization using soft computing approaches. Archives of Computational Methods in Engineering 26(1), 107-118 (2019).

[14] Dewil, R., Küçükoğlu, İ., Luteyn, C., Cattrysse, D.: A critical review of multi-hole drilling path optimization. Archives of Computational Methods in Engineering 26(2), 449-459 (2019).

[15] Ramli, K. D. N. R.: Application of artificial intelligence methods of tool path optimization in CNC machines: A review. Research Journal of Applied Sciences, Engineering and Technology 8(6), 746-754 (2014).

[16] Lai, Y. L., Shen, P. C., Liao, C. C., Luo, T. L.: Methodology to optimize dead yarn and tufting time for a high performance CNC by heuristic and genetic approach. Robotics and Computer-Integrated Manufacturing 56, 157-177 (2019).

[17] Kucukoglu, I., Gunduz, T., Balkancioglu, F., Topal, E. C., Sayim, O.: Application of precedence constrained travelling salesman problem model for tool path optimization in CNC milling machines. An International Journal of Optimization and Control: Theories & Applications (IJOCTA) 9(3), 59-68 (2019).

[18] Guo, E. T., Wu, T., Zhang, L. B., Huang, F. L.: Study on the path optimized method based on an improved clustering ant colony algorithm for CNC laser drilling. In Applied Mechanics and Materials 556, 4439-4442 (2014).

[19] Karuppanan, B. R. C., & Saravanan, M.: Optimized sequencing of CNC milling toolpath segments using metaheuristic algorithms. Journal of Mechanical Science and Technology 33(2), 791-800 (2019).

[20] Essink, W. P.: CNC Milling Toolpath Generation Using Genetic Algorithms. Doctoral dissertation, University of Bath (2016).

[21] Abbas, A. T., Hamza, K., Aly, M. F.: CNC machining path planning optimization for circular hole patterns via a hybrid ant colony optimization approach. Mechanical Engineering Research 4(2), 16 (2014).

[22] Abd Elkhalek Sh, M., Elhakim, M.A., Moughith. W.S.: Optimization of the Idle Path in Drilling on CNC Machining Centre using the GA. In Proceeding of the 12th AMME Conference 16, 18 (2006).

[23] Saravanan, M.: Tool Path optimization by Genetic algorithm for Energy Efficient Machining (2018).

[24] Gupta, A. K., Chandna, P., Tandon, P.: Hybrid genetic algorithm for minimizing non productive machining time during 2.5D milling. International Journal of Engineering, Science and Technology 3(1) (2011).

[25] Oysu, C., Bingul, Z.: Application of heuristic and hybrid-GASA algorithms to tool-path optimization problem for minimizing airtime during machining. Engineering Applications of Artificial Intelligence 22(3), 389-396 (2009).

[26] Montiel-Ross, O., Medina-Rodriguez, N., Sepulveda, R., Melin, P.: Methodology to optimize manufacturing time for a CNC using a high performance implementation of ACO. International Journal of Advanced Robotic Systems, 9(4), 121 (2012).

[27] Rao, R.V, Savsani, V., Vakharia, D.P.: Teaching–Learning-Based Optimization: a novel method for constrained mechanical design optimization problems, Computer-Aided Design 43(3), 303-315 (2011).

[28] Rao, R.V., Patel, V.: An elitist Teaching-Learning-Based Optimization algorithm for solving complex constrained optimization problems. International Journal of Industrial Engineering Computations 3, 535–560 (2012).

[29] Rao, R.V., Patel, V.: Comparative performance of an elitist Teaching-Learning-Based Optimization algorithm for solving unconstrained optimization problems. International Journal of Industrial Engineering Computations 4, 29–50 (2013).

[30] Ebraheem, M., Jyothsna, T.R.: Comparative performance evaluation of Teaching Learning Based Optimization against genetic algorithm on benchmark functions. In: 2015 Power, Communication and Information Technology Conference (PCITC), 327-331 (2015).

[31] Shah, S.R., Takmare, S.B.: A Review of Methodologies of TLBO Algorithm to Test the Performance of Benchmark Functions. Programmable Device Circuits and Systems 9(7), 141-145 (2017).

[32] Wu, L., Zoua, F., Chen, D.: Discrete Teaching-Learning-Based Optimization Algorithm for Traveling Salesman Problems. In: MATEC Web of Conferences 128, 02022. EDP Sciences (2017).

[33] Universität Heidelberg. Institut für Informatik. TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, last accessed 2020/25/05.