A COMPARATIVE STUDY OF METAHEURISTICS APPLIED TO TROUBLESHOOTING OPTIMIZATION PROBLEMS

Lucas Sousa de Oliveira
Instituto Tecnológico de Aeronáutica - ITA
Praça Marechal Eduardo Gomes, 50, São José dos Campos, SP, Brasil, 12228-900
lsoliveira459@gmail.com

Leonardo Ramos Rodrigues
Instituto de Aeronáutica e Espaço - IAE
Praça Marechal Eduardo Gomes, 50, São José dos Campos, SP, Brasil, 12228-904
leonardolrr@iae.cta.br

Takashi Yoneyama
Instituto Tecnológico de Aeronáutica - ITA
Praça Marechal Eduardo Gomes, 50, São José dos Campos, SP, Brasil, 12228-900
takashi@ita.br

ABSTRACT

Metaheuristics are powerful in that they efficiently address complex, real-life-sized problems. Many algorithms have been proposed in the last few years, each one with different features, but few stand out. The purpose of this paper is to compare the recently proposed Teaching-Learning Based Optimization (TLBO) with two of the most well-known methods, namely Simulated Annealing (SA) and Genetic Algorithm (GA). Numerical experiments were conducted using three different instances of the troubleshooting optimization problem with different complexity levels. A comparison among the algorithms was made in terms of convergence speed, accuracy and precision. Algorithm complexity was also taken into account, in order to make the comparison useful for practical application. A thorough analysis was made for each algorithm, considering different perspectives of the data collected during the experiments. The final results showed that TLBO has advantages over the others.

KEYWORDS. Metaheuristics. Troubleshooting Problems. TLBO.

Paper topics: MH - Metaheuristics; OC - Combinatorial Optimization

1. Introduction

Metaheuristics are powerful in the way they efficiently provide good solutions for real-world problems [Simon, 2013]. They have been successfully used when the search space is too large, the problem is too complex, the evaluation function is too complicated, the solution is too constrained, or there is no hint of how to solve the problem [Michalewicz and Fogel, 2004]. A metaheuristic can be defined as a generalization of a heuristic that can be applied to a wide variety of problems. The growing interest in metaheuristic algorithms is due to their flexibility (which allows them to be used for a great variety of problems) and good performance (which allows them to efficiently solve big-size instances of problems).

Metaheuristics suffer from their generality: they lack restrictions on their parameters. Therefore, it is necessary to finely tune the parameters such that computational resources and algorithm accuracy are not affected. The Genetic Algorithm (GA), for example, has a high sensitivity to mutation probability, crossover probability and population size [Pinel et al., 2012; Srinivas et al., 2014].

The selection of good parameters is usually done using parameter tuning or parameter control [Eiben and Smit, 2012]. The former denotes the set of methods that define the parameters before the optimization is run, while the latter denotes the methods in which the parameters are dynamically defined during the execution of the optimization algorithm. Parameter control methods can still be categorized into deterministic, adaptive, or self-adaptive.

A set of good parameters that provides good results for a specific problem is not guaranteed to provide good results for other problems. Parameters are usually mutually sensitive and very dependent on the models and problems at hand. Those reasons strengthen the need for parameter control, even though it is harder to achieve [Wolpert and Macready, 1997].

The Teaching-Learning Based Optimization (TLBO) algorithm is a novel metaheuristic that has been recently proposed by Rao et al. [2011]. It has the population size as its unique parameter, besides the stop criteria. TLBO is based on the teaching-learning process observed in a classroom and simulates the influence of a teacher on the output of a group of students in a class.

TLBO has achieved remarkable performances in different types of problems such as constrained [Rao and Patel, 2012], and unconstrained [Rao and Patel, 2013] optimization problems. It has also been successfully used in combinatorial optimization problems such as the flow shop and the job shop scheduling problems [Baykasoglu et al., 2014] and the set covering problem [Crawford et al., 2015].

In this paper, we compare the performance of TLBO with the performance of Simulated Annealing (SA) and Genetic Algorithm (GA) in terms of convergence speed, accuracy and precision. All the algorithms are used in their original form to provide a fair comparison.

Numerical experiments are conducted using the troubleshooting optimization problem. Troubleshooting optimization problem is a combinatorial optimization problem which is known to be NP-hard for most of real applications [Vomvelá, 2003]. Three troubleshooting models with different complexity levels are used.

The remaining sections of this paper are organized as follows. Section 2 describes the troubleshooting problem. Section 3 presents the basic principles of the metaheuristc algorithms used. Section 4 presents the proposed metric used to compare the results. Section 5.1 presents the results observed during the numerical experiments. Concluding remarks are presented in section 6.

2. Troubleshooting Optimization Problems

Troubleshooting is the name given for the sequence of actions performed in order to fix a given system. These actions are commonly separated into two categories: diagnostic and repair [Vomvelá, 2003].

The first type of action, usually less expensive and time consuming, can be inconclusive, i.e., represent some cost for no advancement toward an actual solution. Repair actions, on the other hand, are a step in the direction of solving the problem. Several actions might be required to fix a single fault, as well as several faults might have occurred simultaneously.

These types of problems are very practical and common in our daily lives. The decision-theoretic troubleshooting comes as a way to model and study our way of making decisions to optimize the troubleshooting process and thus minimize the costs involved.

A common way to represent this problem is through oriented graphs, such as the one presented in Figure 1. Nodes represent the possible failure modes (F_i), diagnose questions (Q_i) and repair actions (A_i), while the oriented vertices represent the sequence of actions that can be executed [Vianna et al., 2016].

Costs are represented by C_i, while cost clusters are represented by K_i. A fault caused by failure mode F_i is only fixed when all repair actions connected to it are executed. A diagnostic question isolates a subset of possible failure modes. Each action and question has an associated cost, which is incurred if the action or question is executed. A cluster cost is incurred if at least one action or question connected to it is executed.

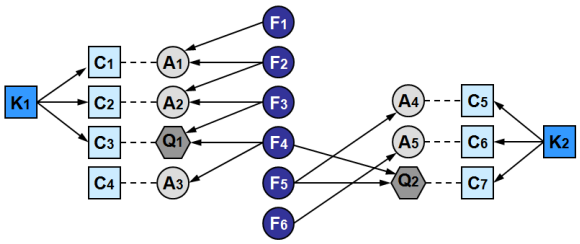


Figure 1: Troubleshooting Model 1 - Bayesian network representing a simple troubleshooting problem.

Several assumptions need to be made to ensure optimality of the sequence found [Langseth and Jensen, 2001]:

- **Single Fault:** only one fault can be present at the system at a time.
- **Perfect Repair:** repair actions are always effective.
- **Fixed Cost:** costs do not vary with time.
- **No questions:** diagnose actions are not allowed.
- **Independent Actions:** each action addresses exactly one fault.

Without these restrictions there is no polynomial approximation algorithm for such problem [Lín, 2014].

As mentioned earlier, every troubleshooting strategy has an associated Expected Cost of Repair (ECR). This cost is calculated in terms of the action costs C_i, the cluster costs K_i and the failure mode probabilities $P(F_i)$. The ECR is defined as showed in Equation (1).

$$\text{ECR}(s) = \sum_{i=1}^{n} \left(P(F_i) \cdot \sum_{j \in l_{F_i}} C(A_j) + C(K_j) \right) \tag{1}$$

where $P(F_i)$ is the probability of occurrence of F_i, n is the number of possible failure modes, $C(A_j)$ with $j \in l_{F_i}$ is the cost of each action (repair or diagnose) and its cluster in the path l_{F_i}, l_{F_i} is the path to repair F_i according to the strategy s. This will be used as the objective function of the metaheuristic optimization algorithm. Clusters are activated only once for every set of actions that use them.

3. Metaheuristics

3.1. Teaching-Learning-Based Optimization

The TLBO algorithm is divided into two main parts: the Teacher Phase and the Student Phase, which is also known as the Learner Phase [Rao and Patel, 2013]. During the Teacher Phase, students learn from the teacher, while in the Learner Phase students learn through the interaction among themselves.

There is a solution X associated with each student, which corresponds to a possible solution to the optimization problem under consideration. Also, there is a result $f(X)$ associated with each solution (or student), which can be obtained by evaluating the solution X using the objective function f. In the troubleshooting problem considered in this paper, a solution X corresponds to a troubleshooting strategy S and the associated result $f(X)$ corresponds to its Expected Cost of Repair, denoted by $ECR(S)$ [Vianna et al., 2016].

3.1.1. Teacher Phase

In this phase, the algorithm simulates the learning of the students from the teacher (best solution). During this phase, the teacher makes an effort to increase the mean result of the class.

Consider a group of n students. Let M_i be the mean solution of the students and T_i be the teacher at iteration i. The teacher T_i will make an effort to move M_i to its own level. Knowledge is gained based on the quality of the teacher and the quality of students in the class. The difference D_i between the solution of the teacher, X_{Ti}, and the mean solution of the students, M_i, can be expressed according to Equation (2):

$$D_i = r_i(X_{Ti} - T_F \cdot M_i) \tag{2}$$

where r_i is a random number in the range $[0, 1]$ for iteration i and T_F is a teaching factor for iteration i, which is randomly set to either 1 or 2 according to Equation (3):

$$T_F = round(1 + rand(0,1)) \tag{3}$$

Based on the difference D_i, the existing solution of student k in iteration i, X_{ki}, with $k \in \{1,2,\ldots,n\}$, is updated in the teacher phase according to Equation (4):

$$X^{\star}_{ki} = X_{ki} + D_i \tag{4}$$

where $X^{\star}_{ki}$ is the updated value of X_{ki}.

If $f(X^{\star}_{ki})$ is better than $f(X_{ki})$, $X^{\star}_{ki}$ is accepted and replaces X_{ki}. Otherwise, $X^{\star}_{ki}$ is discarded.

3.1.2. Student Phase

In this phase, the algorithm simulates the learning of the students through interaction with one another. During this phase, students gain knowledge by discussing with another students who have more knowledge [Rao and Patel, 2013].

Consider a pair of students y and z. Let X_{yi} and X_{zi} be the solutions of students y and z at iteration i, respectively. If $f(X_{yi})$ is better than $f(X_{zi})$, the solution of student z is updated according to Equation (5). Then, $X^{\star}_{zi}$ will replace X_{zi} if $f(X^{\star}_{zi})$ is better than $f(X_{zi})$.

Similarly, if $f(X_{zi})$ is better than $f(X_{yi})$, the solution of student y is updated according to Equation (6). Then, $X^{\star}_{yi}$ will replace X_{yi} if $f(X^{\star}_{yi})$ is better than $f(X_{yi})$.

$$X^{\star}_{zi} = r_i(X_{yi} - X_{zi}) \tag{5}$$
$$X^{\star}_{yi} = r_i(X_{zi} - X_{yi}) \tag{6}$$

At the end of each iteration, the stop criteria must be checked. Different stop criteria may be adopted. Some of the most commonly adopted stop criteria are the maximum number of iterations, the maximum number of successive iterations without any improvement and the maximum simulation time. In this paper, the maximum simulation time is adopted as the stop criterion.

3.2. Simulated Annealing

The Simulated Annealing (SA) algorithm was developed by Khachaturyan et al. [1979] and uses the annealing process present in metallurgy to improve the traditional greedy search. Besides simple, SA is very fast and can be applied to both continuous and combinatorial applications. It is commonly used when finding a good solution fast is essential, but finding the optimal solution is not.

The SA algorithm is mainly divided into three blocks: temperature management, neighborhood generation and acceptance criteria. These blocks are briefly described in the following lines.

3.2.1. Temperature Management

The SA algorithm has three important temperatures that must be defined: the initial temperature, denoted by T_0, the current temperature, denoted by T_i (where i is the current iteration), and the final temperature, denoted by T_F (where N is the maximum number of iterations).

The behavior of T_i can follow any profile desired, but it is typically chosen to be a monotonic descending function. This paper uses a linear function to describe the temperature profile, as shown in Equation (7).

$$T_i = (T_0 - t_{i-1})\frac{T_0 - T_F}{t_F} \tag{7}$$

where t_{i-1} is the time since the beginning of the simulation until the start of the current iteration, and t_F is the expected final simulation time.

The stop criteria is also attributed to this block because when $t_i > t_F$ the simulation ends.

3.2.2. Neighborhood Generation

The neighborhood generation block can assume different forms. The only requirement is the capability of providing a solution vector slightly different from the previous one.

In the troubleshooting optimization problem considered in this paper, a candidate solution is a permutation of the available repair actions A_i and diagnostic questions Q_i. Thus, new individuals are generated by swapping the position of two elements in the original solution.

3.2.3. Acceptance Criteria

The acceptance criteria in SA governs its ability to, from time to time, accept a solution that is worse than the previous one. In this paper, the acceptance criteria is based on the commonly adopted Metropolis' Algorithm [Metropolis et al., 1953]. For minimization problems, the probability of a solution to be accepted is defined according to Equation (8).

$$P_A = \begin{cases} 1 & \text{if } C_i < C_{i-1} \\ e^{\frac{(C_i)-(C_{i-1})}{T_i}} & \text{if } C_i > C_{i-1} \end{cases} \tag{8}$$

where C_i is the cost of the current solution, C_{i-1} is the cost for the previous solution, T_i is the temperature for the current iteration.

3.3. Genetic Algorithms

Genetic Algorithms (GA) are one of oldest class of metaheuristics and one of the first Evolutionary Algorithms that became available. GA were introduced by Holland [1975]. Genetic Algorithms use concepts observed in human genetics to combine, mutate and select chromosomes until a stop criteria is met. They have been used successfully in both continuous and combinatorial optimization, and their success is mainly attributed to their simplicity and flexibility.

The first step required by the algorithm is the definition of a way to encode the solution vector into a chromosome, which is composed by genes. A collection of chromosomes forms the population in the algorithm. In this paper, the chromosome is a sequence of repair actions and diagnose questions identified by non-repeating integer numbers.

The basic algorithm proposed by Holland takes the population of chromosomes and passes it through a series of five stages: initialization, evaluation, selection, crossover and mutation. A brief description on each stage is presented in the following lines.

3.3.1. Initialization

The initialization stage defines a way to generate a valid initial population. In this paper, the initial population was built randomly.

3.3.2. Evaluation

The evaluation stage is highly dependent of the problem at hand since it attributes a score to each of the individuals in the population. It is also usually delegated to this stage the control of the stop criteria. The objective function used in this paper is the Expected Cost of Repair (ECR) defined in Equation (1).

3.3.3. Selection

The selection stage mimics the natural selection (or survival of the fittest) process, in which only the strongest individuals survive. This process receives the score from the evaluation stage in order to properly select the individuals.

There are several methods available to implement this stage. They can be classified as deterministic, stochastic or hybrid. Deterministic methods are more well-behaved and easier to debug. The stochastic methods mimic the natural selection process more ideally, but tend to be less well-behaved. The hybrid approaches are usually preferred because they tend to present the benefits of the previous methods without their disadvantages.

In this paper, we used a hybrid approach in which a few elite individuals survived between generations while others were picked randomly. Non-elite individuals were selected based on a rank probability, in which they are ranked according to their fitness. The probabilities assigned depend on a given P_r. Each individual receives a probability P_i according to Equation (9).

$$P_i = P_r \cdot (1 - P_r)^i \tag{9}$$

where i is the rank for the solution.

3.3.4. Crossover

The crossover stage mimics the reproduction process, in which the genes from two parents are mixed to form new individuals. This process usually takes a probability P_C, also known as crossover rate, to determine if the parents will be crossed or kept.

Several implementations are available to implement this stage. In this paper, we use the Ordered Crossover (OX) method. The OX method starts by randomly defining two cut points. Then, the parents are broken into three sections. A queue is then built by selecting, in this order, the second section of the first parent, the second section of the second parent, the third section of the second parent and the first section of the second parent. The queue is then scanned for duplicated and they are removed leaving only the first occurrence.

3.3.5. Mutation

The mutation stage mimics the mutation process that occurs with the genes in our cells. It allows us to evolve from one generation to another and is considered to be one of the main reasons for the diversity of the species. The individuals in the population are mutated according to a mutation probability P_M.

4. Metric

This section presents the metric proposed to compare the performance of the different optimizations for solving the troubleshooting optimization problem. In each simulation, the following outputs are recorded:

- the Estimated Cost of Repair (ECR), which indicates the quality of a given solution;
- the iteration time, which indicates the exact time it took for every iteration to run;
- the iteration count, which indicates how many iterations occurred before the current one; and
- the solutions vector for every iteration.

A simple metric, named Score and denoted by S, is proposed to evaluate the performance of the algorithms. The computation of S for continuous and discrete problems are presented in Equations (10) and (11), respectively.

$$S = \int_0^{t_{max}} ECRdt \tag{10}$$

$$S \approx -\sum_1^N \left(\min(ECR_i, ECR_{i-1}) + \frac{|ECR_i - ECR_{i-1}|}{2} \right) \cdot (t_i - t_{i-1}) \tag{11}$$

where ECR_i is the Expected Cost of Repair in iteration i, t_i is the simulation time at iteration i.

5. Numerical Experiments

A limitation faced when performing such simulations was the restricted computational time available, a restriction is often observed in commercial applications. One consequence of this restriction is that it is not viable to sample the parameter space extensively, and so a subset of conditions will have to be studied and the conclusions will have to be extrapolated. Design of Experiments (DOE), which is a procedural way of analyzing some experiment, was used to define few experiments that would help map the parameter space and help draw conclusive results.

A simple algorithm derived from the steepest descend was used to guide the DOE through the parameter space exploration. Each algorithm had a initial parameter setting arbitrarily chosen according to current best practices. Full-factorial form, which uses 2^k points where k is the amount of parameters defined a Region of Interest (ROI) to be analyzed, was used since aliasing (using 2^{k-n}) would reduce the statistical significance of the data. As an optimization algorithm itself, the stop criterion used for the DOE was based on signal-to-noise ratio concept, where the algorithm would stop if $\sigma/\mu > 1$ for S or if 20 DOE iteration were executed.

A Monte Carlo approach was also used. The simulations conducted in this paper were repeated 5 times for each parameter/algorithm setting. This value was empirically set in order for the whole simulation to be completed in approximately one full day, as shown in Table 1.

Table 1: Complete simulation time estimate

	TLBO	SA	GA
Number of Models		3	
Maximum DOE Iterations		20	
# Input Parameters	1	2	5
DOE Experiments per Iteration	2	4	32
Fixed Simulation Time (Seconds)		10	
Estimated Simulation Overhead		10 %	
Estimated Time (Hours)	0.4	0.7	5.9
Total Estimated Time (Hours)		**7.0**	
Monte Carlo Repetitions		5	
Estimated Time (Hours)	1.8	3.7	29.3
Total Estimated Time (Hours)		**34.8**	

The algorithms were implemented in Matlab®R2016a and ran on a HP ENVY dv6-7200 CTO with Intel Core i7-3630QM CPU @ 2.40 GHz, 12 GB of RAM, Nvidia 920M GPU with 2 GB of dedicated memory and Windows 10 Pro OS.

5.1. Simulation Results

The results were measured in terms of speed, accuracy and precision, each calculated over a dimension of the data collected here. Accuracy was measured by how far the final values were from the best solution found between all the algorithms. Precision was measured by the standard deviation of the value in the best experiment performed. Speed was calculated directly from the ECR by comparing the time taken for the algorithm the 105% of the best solutions found. The best solutions found had ECRs equal to 78.5, 199.0 and 206.3 for Models 1, 2 and 3 respectively.

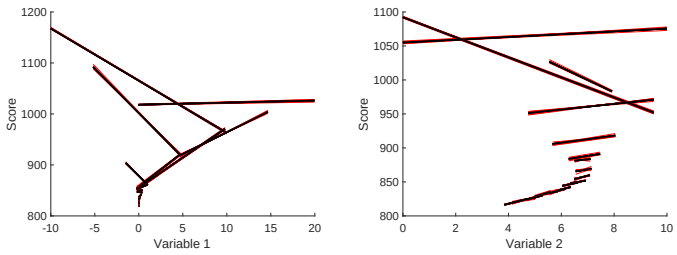
5.1.1. Simulated Annealing

A initial ROI was set, as hard limits to encapsulate the ROIs, is shown in Table 2. These values where chosen after a few experiments that indicated they would perform fairly for the problem at hand. The hard limits, values for which the ROIs would be truncated, were defined by the simulated annealing intrinsic features.

Table 2: Initial Region of Interest and Hard Limits for the SA algorithm.

Parameters	Initial Region		Hard Limits	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
T_0	10	50	Unconstrained	Unconstrained
ΔT	0	10	0	Unconstrained

Figure 2 shows an example of the simulations performed for Model 1. The results are summarized in Table 3. These simulation were unique in that they did not reach the stop criterion for two of the three models, which happened because the standard deviation would decrease as much as the mean slope. The SA reached the best mean score for Model 3, even though its precision dropped considerably. Although its score was the best for Model 3, its best solution did not reach the 5% value above the best solution (216.6 for Model 3) to be considered fast.



(a) Model 1, T_0 (b) Model 1, ΔT

Figure 2: Example of slopes of each ROI as the DOE seached for a better parameter set for Model 1. Black thick lines represent the mean slope, while the red thin line represents the slope for each Monte Carlo repetition of the ROI gradient.

Table 3: Final simulation results for SA algorithm.

Parameter	Model		
	1	2	3
T_0	0,00	-0,02	0.10
ΔT	3.54	4.67	4.80
Best Mean Score	812.67	2080.72	2245.93
Max DOE Iteration	Max	Max	12
Accuracy (% Worse than Best Solution)	3.5%	4.6%	8.9%
Precision (Standard Deviation as %)	0.1%	0.2%	1.0%
Speed (5% to Best, in seconds)	0.06	1.23	≥ 10

5.1.2. Genetic Algorithm

The initial ROI and its hard limits is shown in Table 2. These values where chosen after a few experiments that indicated they would perform fairly for the problem at hand. The hard limits, values for which the ROIs would be truncated, were defined by the Genetic Algorithm intrinsic features. The lower population bound, for example, was chosen because with less than 2 individuals there would not be enough parents to crossover.

Table 4: Initial Region of Interest and Hard Limits for the GA algorithm.

Parameters	Initial Region		Hard Limits	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Population Size	10	50	2	Unconstrained
Mutation Rate	10%	20%	0%	100%
Crossover Rate	10%	20%	0%	100%
Rank Probability	10%	20%	0%	100%
Proportion of Elite Individuals	50%	60%	0%	100%

The simulation summary is presented in Table 5. GA presented the slowest, less precise and less accurate performance among the three algorithms. Models greater than Model 1 could not have its minimum value found in the appointed time. Its accuracy deviation increased exponentially as the models got more complex. Its speed was also very poor, making it not able to reach the best solution in the appointed time.

Table 5: Final simulation results for GA algorithm.

Parameter	Model		
	1	2	3
Population Size	217	179	134
Mutation Rate	41.35%	43.10%	41.73%
Crossover Rate	19.92%	21.15%	16.57%
Rank Probability	2.13%	0.36%	2.16%
Proportion of Elite Individuals	75.07%	74.97%	66.03%
Best Mean Score	808.4	2268.57	2895.92
Max DOE Iteration	14	13	10
Accuracy (% Worse than Best Solution)	3.0%	14.0%	40.4%
Precision (Standard Deviation as %)	0.9%	1.5%	7.3%
Speed (5% to Best, seconds)	1.6987	≥ 10	≥ 10

5.1.3. TLBO

The initial ROI and Hard Limit chosen for TLBO are those shown in Table 6. The lower bound was chosen because 2 individuals because there needs to be a learner in the algorithm besides the teacher.

Table 6: Initial Region of Interest for the TLBO algorithm

Parameters	Initial Region		Hard Limits	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Population Size	2	10	2	Unconstrained

The Table 7 summarizes the results for the TLBO simulations. Its accuracy, precision and speed equals the SA algorithm for the smaller models, but its accuracy and precision gets slightly worse than SA for Model 3.

Table 7: Final simulation results for TLBO algorithm.

Parameter	Model		
	1	2	3
Population Size	11	8	2
Best Mean Score	785.22	2008.37	2297.13
Max DOE Iteration	1	1	1
Accuracy (% Worse than Best Solution)	0.0%	0.9%	11.3%
Precision (Standard Deviation as %)	0.0%	1.0%	2.2%
Speed (5% to Best, seconds)	0.03	2.46	9.25

6. Conclusion

This work was set out with the purpose of comparing metaheuristic optimization algorithms to the newly proposed Teaching-Learning Based Optimization. To do so, the Troubleshooting problem was coded and simulated within a common-ground framework. This comparison was

made pragmatically and focused on convergence speed, accuracy and precision. The algorithms complexity was also taken into account, in order to make the comparison useful for practical application.

All three algorithms are considered to be conceptually simple, but they allow a lot of design choices that can complicate optimization task. GA was the method with the most choices to be made which, summed to its highly sensitive parameter tuning, made it barely practical. SA and TLBO presented the opposite characteristic: they seemed very practical to set. Even though SA has one parameter more than TLBO, it was not hard to set it to a good value. One preoccupation towards SA is that its Metropolis' Criterion can make the algorithm perform very poorly if not properly contained, tending to a Random Search.

TLBO was more accurate for Model 1 but quickly got worse for Model 3. SA's was more accurate for Model 3. The accuracy of GA got a lot worse as the model became more complex.

Although TLBO and SA present the same level of precision for small models, SA performed better for Model 3. One point to note is that TLBO took very few DOE iteration to tune. GA presented a good precision until the model became too complex.

All the algorithms studied are very fast converging. GA and SA could not compute Model 3 in time, even though SA presented the best mean score of all algorithms. TLBO was the fastest algorithm.

Potential extensions of this work would be to allow the algorithms to explore the parameter space more extensively and to compare variants of these same algorithms. One other interesting extension would be to combine these algorithms to get the best of each. By taking a solution quickly found by TLBO and providing it to SA to improve its precision or to GA to increase the space exploration, the resulting algorithm could present even better performances.

References

Baykasoglu, A., Hamzadayi, A., and KÖSE, S. Y. (2014). Testing the performance of teaching learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. *Information Sciences*, 276:204–218.

Crawford, B., Soto, R., Leiva, F. A., Johnson, F., and Paredes, F. (2015). Problema del conjunto de cobertura resuelto mediante el algoritmo binario de optimización basado en enseñanza-aprendizaje. In *Conferência Ibérica de Sistemas e Tecnologias de Informação*, p. 106–109, Águeda. IEEE.

Eiben, A. E. and Smit, S. K. (2012). Evolutionary algorithm parameters and methods to tune them. In Hamadi, Y., Monfroy, E., and Saubion, F., editors, *Autonomous Search*, p. 15–36. Springer Berlin Heidelberg, Berlin, Heidelberg.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 2 edition.

Khachaturyan, A., Semenovskaya, S., and Vainshtein, B. (1979). Statistical-thermodynamic approach to determination of structure amplitude phases. p. 519–524.

Langseth, H. and Jensen, F. V. (2001). Heuristics for two extensions of basic troubleshooting. *Frontiers in Artificial Intelligence and Applications*, 66:80–89.

Lín, V. (2014). Decision-theoretic troubleshooting: Hardness of approximation. *International Journal of Approximate Reasoning*, 55(4):977–988. Special issue on the sixth European Workshop on Probabilistic Graphical Models.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087.

Michalewicz, Z. and Fogel, D. B. (2004). *How to Solve It: Modern Heuristics*. Springer, enlarged 2nd edition.

Pinel, F., Danoy, G., and Bouvry, P. (2012). Evolutionary algorithm parameter tuning with sensitivity analysis. In *Proceedings of the 2011 International Conference on Security and Intelligent Information Systems (SIIS'11)*, p. 204–216, Berlin, Heidelberg. Springer-Verlag.

Rao, R. V. and Patel, V. (2012). An elitist teaching-learning based optimization algorithm for solving complex constrained optimization problems. *International Journal of Industrial Engineering Computations*, 3:535–560.

Rao, R. V. and Patel, V. (2013). An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems. *Scientia Iranica*, 20:710–720.

Rao, R. V., Vakharia, D. P., and Savsani, V. J. (2011). Teaching-learning-based optimization:a novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43: 303–315.

Simon, D. (2013). *Evolutionary Optimization Algorithms*. Wiley. URL `https://books.google.com.br/books?id=gwUwIEPqk30C`.

Srinivas, C., Reddy, B. R., Ramji, K., and Naveen, R. (2014). Sensitivity analysis to determine the parameters of genetic algorithm for machine layout. *Procedia Materials Science*, 6:866–876. 3rd International Conference on Materials Processing and Characterisation (ICMPC 2014).

Vianna, W. O. L., Rodrigues, L. R., Yoneyama, T., and Mattos, D. I. (2016). Troubleshooting optimization using multi-start simulated annealing. In *10th Annual IEEE Systems Conference*, Orlando, FL, USA.

Vomvelá, M. (2003). Complexity of decision-theoretic troubleshooting. *International Journal of Intelligent Systems*, 18(2):267–277.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82.