 doi:10.20944/preprints201611.0033.v1

Article

Construct Linear Polynomial Complementary Transformation for NP-Completeness Using Parallel Genetic Algorithm

Tarik Eltaeib * and Julius Dichter

Department of Computer Science, University of Bridgeport, 126 Park Ave, Bridgeport, CT 06604, USA; dichter@bridgeport.edu

* Correspondence: teltaeib@bridgeport.edu; Tel.: +678-237-6229

Abstract: This paper examines the correlation between numbers of computer cores in parallel genetic algorithms. The objective to determine the linear polynomial complementary equation in order represent the relation between number of parallel processing and optimum solutions. Model this relation as optimization function (f(x)) which able to produce many simulation results. F(x) performance is outperform genetic algorithms. Compression results between genetic algorithm and optimization function is done. Also the optimization function give model to speed up genetic algorithm. Optimization function is a complementary transformation which maps a TSP given to linear without changing the roots of the polynomials.

Keywords: genetic algorithms; parallel computation; computational complexity; algorithms; optimization techniques; traveling salesman problem; NP-Hard problems; Berlin-52 data set; machine learning; linear regression

1. Introduction

NP problems and NP-completeness problems are nondeterministic polynomial problems. The main difference between the NP and NP- completeness is that in NP problems the solution is verifiable unlike NP completeness problems. In other words, the complexity of those problems cannot be bounded, since the polynomial algorithms are unknown within NP- completeness. However, there are alternative algorithms that solve NP problems[1]. A famous one is referred as the Genetic Algorithm (GA).The GA has long been used as a heuristic search technique[2]. This technique serves as a probabilistic search that applies natural phenomena to find optimum solution[3]. In NP-completeness problems the GAs are determines the optimum solution[4]. Since NP and NP-completeness are hard to solve in a polynomial time algorithm, GA is used to determine an optimum solution. Furthermore, the results of applied GA with NP-completeness problem cannot be verified[4].

In our research, we focused on the Traveling Salesperson Problem (TSP) problem, which is classified as an NP-completeness problem[5]. TSP problem is given a set of cities. The salesman has to visit each city only once and returning to the starting city. The problem of traveling salesman wants to minimize the total length of the tour[6].

Since the TSP is difficult to solve in a polynomial algorithm, we applied a GA to determine an optimum solution. However, GA results cannot be verified, since the TSP is NP-completeness problem[7, 8]. In conjunction with TSP, the GA created a population based on the theory of fitness evolution[9, 10]. Therefore, GA consists of the guessing stage and the checking stage. Additionally, there are several parameters that controlled the performance of GA[11]. Such as, population size, crossover probability, and mutation probability and all factored into the GA's results. Although GA is widely acceptable technique with NP and NP-completeness, having several drawbacks. The first drawback is that the GA does not have a concrete initial population; the GA often trapped in local

 doi:10.20944/preprints201611.0033.v1

minimum[12]. If initial chosen population is not good, it convert hard to find the correct solution of the problem[13].

The second drawback accused in the evolution stage because solid criteria was missing that could evaluate populations. The problem in the evolution stage, the population could evaluate either against the unknown environment or the evolution function complexity is running in exponential time[14].

Genetic algorithm is fit for parallel execution which increases the speed of search. GA can run by distributing over a number of CPUs. The parallel application must run on computer architecture that supports multi-threads, simultaneously, such as multiple instruction stream data stream (MIMD) otherwise known as multi-cores computers. Intuitive numbers of computer cores play a very important role in determining the optimum solutions[15]. In our experiment we built application that applied multithread techniques .Then we ran our application in different number of threads. For example when we ran two threads we called number of GAs is two, and three threads will have three GAs and so on.

In our work, we utilize the multi-core architecture by using parallel processing to gain insight into linear effects of changing the TSP solutions to CPU core allocation.

We conducted experimental research to determine the following: What the interaction between number of computer cores and TSP? In particular, we explored the correlation between the increased number of parallel processing which mean number of GAs and finding better optimum solution in TSP. Moreover, we explored how we can transfer NP-completeness problem to be NP problem using this correlation. By demonstrating this we captured a crucial properties in order to find a complementary linear polynomial function. This function works as a polynomial transformation mapping model for NP-completeness to NP problem. We showed how to use this derivative polynomial function to predict the list of optimum solutions.

2. Supervised machine learning

Supervised machine learning (SML) using regression is well-established method in data analysis[16]. The main purpose of using regression model is to find the relation between two independent variables[17]. In our work, the machine learning algorithm will be run in the beginning to create its training dataset. While the algorithm to build its training set, it can use it to specify the best number of assigned cores for this specific problem that run on that particular system.

We presented how we built training data set and how we used it formulate the relation between number of GAs and CPU core allocation. Finally, analyze this mathematical model to determine how we can construct it as polynomial formula.

In machine learning there are many techniques such as linear regression and random forest. In our research, we used regression model to formulate the relation between Number of GAs and finding optimum solution. We build our dataset for two parameters that number of GAs and corresponding optimum solution. We used linear regression model to build training set to build up predicator in real dataset to find what the optimum number of GAs values that required for this dataset in a parallel environment to turn quickly to an acceptably good target solution.

3. Genetic Algorithms

The main reason to use the Genetic Algorithms (GAs) to find the global optimum; for that GAs are technique for solving NP problems which their growth exponential[18]. GAs involve in developing a population of individuals. GA are population established optimization procedures intended for searching optimal solutions in complex spaces. GAs are mimic biological processing in nature in order to get better populations. These algorithms are mimic on some biological procedures that can be gotten in Nature, like natural selection [19]or genetic inheritance [5]. The initial population is made randomly. A fitness evaluation gives a cost to each individual. This assessment can be did by an objective function which call fitness value and it done a mathematical. The stop condition is

 doi:10.20944/preprints201611.0033.v1

typically set to reach a number of repetitions, or to catch a solution to the problem if it is known beforehand.

In general the GAs apply a single population of individuals and manipulate them with different parameters. However, there another type of GAs called decentralized and also it is known as structured GAs. This type is fitting for parallel techniques since the population is not centralized. Each individuals has their fitness so that each one represent possible solutions. Fitness reflect numerical measurement used by GA to guide the search processing. Because the isolated populations are the main aspect of decentralized population that enable to implement the parallel technique in this type smoothly and keep a higher genetic differentiation[20].Moreover, since the enrich and the variety of initial sampling , decentralized genetic algorithms (dGAs) have demonstrated better performance in search space comparing with ordinary genetic algorithms[21].

Cellular genetic algorithms (cGAs) another type which are fitting with parallelism[2]. They are similar to dGAs which worked with isolated population .however, the cGAs used communication utility between neighbored in order to maintain high quality of diversity[22]. Furthermore, the cGAs consist of small neighborhood which only interact with its adjacent neighbors. This technique make the cGAs discovering the search space more effective because they induced spreading of solutions through the population in order to maintain the diversity and intensely for each neighborhood[23].

However, we need balance between the exploration of new area of search space and exploitation of computer resources such as processor. If we able to accomplish this balance that will lead us to high performance of GAs. In fact, this exploration and exploitation can be an impact each other, meaning increasing or decreasing one of them can influence another. Thus, the parallelism is necessary to not only decrease the processing time, but also to improve the quality of solutions. In the beginning we would like to introduce some terminology definition of GAs[14].

4. Experimental Set-Up

We developed framework using Java and our instance for traveling salesperson problem. Our framework consists of a set of threads working in parallel on a multi-core machine solving a single Traveling Salesman person (TSP) optimization problem. First of all, we give brief description of hardware and software architecture. The computer specification that we used to run this experimental is Intel® core 4, speed 2.8 Mhz. Software is 64 Operating system Microsoft windows 10 Pro. Our idea is built around the theory of independent evolution of separate worlds. Each GA solver initially with some random solutions. Since each GA is independent, the solutions will vary, and some GAs will have solutions that are better than others[8]. However, since each solution is a sequence through an entirely linked graph, every solution, even a poor one may have a section which would make an efficient part of a good solution[24]. Even two good solutions might be good for different reasons[25, 26]. One solution might have an efficient solution for one graph section while another good solution could have efficiency in another section[27]. The merging and crossing over of the different solutions is the elementary idea to improvement in GAs solution [28-30]. In our framework, we allow multiple GAs to work concurrently and independently of all other GAs. Note that concurrently may or may not be simultaneously. If there are 8 GAs and 8 CPU cores, then they may work simultaneously. However, with 16 GAs, two GAs would be running on a single CPU core. With 64 GAs, that number would increase to 8. As the number of GAs increases as a multiple of the number of cores. We need find what the optimum number of CPU cores, this ideal point which paly

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2016 doi:10.20944/preprints201611.0033.v1

as crucial parameter as well for finding optimum solution, and this optimum point we called C_i which mean is the number of ideal GAs that we can generated with (i) number of cores. For example if C_i= 64 GAs for 8 CPU. In the beginning, we built date set which providing foundation for understanding GA performance proportion with number of cores.

Thus, we fixed all other GA parameters and we ran 10 times and each time for 30 minutes. Then we store the best optimum results in text file format to use it later in next stage to find correlation between those sets of fitness and number of threads.

4.1. Framework GAs parameters

In our framework we have many parameters as input variables such as Area, target, city count [number of cities] display count, thread count, population size, exchange frequency, number of running, and period of each time. For our experimental we used the benchmark TSP is Berlin52 which can be found in TSPLIB: http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/ TSPLIB95/TSPLIB.html In our framework there are some configuration such as number of running meaning how many times we want to the specific experimental to be run and this feature of automatic running, so we can assign this framework to run for 10 trails end each time run of 30 minutes period. We ran 10 experiments automatic and the results of this experiments will be stored in text files with all information needed. For example, we can run the first experiment with the following parameters as shown in Table [1]

Parameters	Values
Number of City	52
Population size	1640
Display	1,2,3...,10
Number of Run	30
Duration	60 second

[1] Input parameters values

4.2. Building Training Dataset

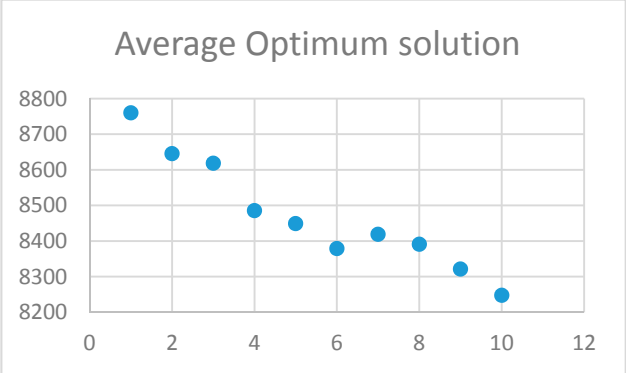
Our framework produces the optimum results as shown in table 2. We have the value of number of threads to be 1, 2, 3,4,5,6,7,8,9 and 10 and we keep all other GAs parameters fixed. Then, we have run for 30 times and each time 1 minute .In every time we get the average of fitness. This is our training dataset that we are going to use it next stage which is observation stage.

No of Threads	Average of Optimum
1	8759.75
2	8645.18
3	8618.24
4	8485.11
5	8448.81
6	8378.67
7	8418.53
8	8390.70
9	8320.90
10	8247.50

[2] List of optimum solutions from framework

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2016 doi:10.20944/preprints201611.0033.v1

Next, we used this dataset to build model mathematically formulae that represent the relationship between the number of threads with those solutions. Then, in proceed used a linear regression to be able to generate quantitative analysis between these two variables. In the begging if we assume $f(x) = C_i$ where x the required number threads to and C_i optimum solutions, and remember f(x) is the mathematical model we are seeking to generate. In this model should able find depend on some variables how many GAs that are required to get the optimum solution in appropriate time an based on number of available CPU core in that specific system. In figure 1 shown the average of optimum solutions and number of threads. It shows there is systemically decremented and their coloration between number of threads and getting optimum solutions.



(**Figure 1 Average of optimum solutions and number of threads.**)

5. Linear Regression Model

In linear regression technique, we placed in scatterplot the number of threads and the average of optimum solutions that we got, and in order to find polynomial formula to demonstrate this relation. From table 2 we have two variables, optimum solution, and number of threads. Finding the mathematical model that represent the relationship between the optimum solution

The basic equation form such as y=mx+b which can represent in our work

as following: $F(x) = C_0 + C_1\ X$, where the *function* $F(x)$ that expresses the potential performance gain when *x* threads run

C_0 it is average of optimum values,

C_1 it is slop of parameters

Then we need to find the best-fitting

Curve for our data set of by using the residuals$\sum(c_i - \hat{c_i})^2$, where c_i is of values whether is fitness or GAs count, $\hat{c_i}$ is the average. Then next step will be slop calculation C_1 :

$$C_1 = \frac{\sum(y_i - \hat{y})(x_i - \hat{x})}{\sum(x_i - \hat{x})^2}$$

$$\sum(x_i - \hat{x})^2 = 85$$

$$\sum(y_i - \hat{y})(x_i - \hat{x}) = -4143.89$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2016 doi:10.20944/preprints201611.0033.v1

$$C_1 \ \frac{85}{-4143.89} = -0.205$$

2nd step we want to calculate C_0

$$C_0 \ = \ y^{\wedge} - \ C_1 \ x^{\wedge}$$

$$C_0 \ = \ y^{\wedge} - 0.205 \ x^{\wedge}$$

$$\because C_1 = 0.205, \quad x^{\wedge}=6, \quad y^{\wedge}=8471.34$$

$$\therefore \ C_0 \ = 8471.34-(0.205\ ) * 6 \qquad = 8470.11 \approx 8470$$

Then next we applied these values for our form

$$\mathrm{F}(x) = C_0 + C_1 \ X$$

$$\mathrm{F}(x) = 8470 - 0.205\, X$$

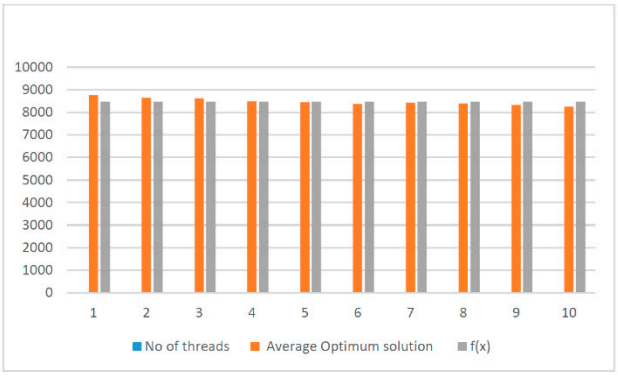
This the constructed polynomial equation that describe the correlation of this problem instance of TSP Berlin 52. After that we are applying this linear equation to predicate solutions .Table 3 shown the results first column X which represent number of threads which is the inputs will be varieties from x=1 to x 10. The second column is representing the optimum solutions from our equation$\mathrm{F}(x) = 8470 - 0.205\, X$.

No of Threads	$\mathbf{F(x) = 8470 - 0.205\, X}$	Average of Optimum
1	8,469.80	8759.75
2	8,469.59	8645.18
3	8,469.38	8618.24
4	8,469.18	8485.11
5	8,468.97	8448.81
6	8,468.77	8378.67
7	8,468.56	8418.53
8	8,468.36	8390.70
9	8,468.15	8320.90
10	8,467.95	8247.50

[3] the optimum solution from f(x)

A column chart is a graphic representation of data. In the chart below figure 2 , the x-axis represents the threads and y-axis represent the average of optimum solutions from framework . From the chart we can clearly see that results of both the software and function f(x) are very close to each other.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2016 doi:10.20944/preprints201611.0033.v1



(a)

Figure 2. This a column chart show comparison between performance of f(x) and framework

6. Empirical Results and discussion

In the table (3) the first Colum represents threads, second column represents results from software and third column represents results from function f(x).we have collected data for 10 threads both form our framework and the function f(x). We used only 10 threads because we can't use more than 10 threads using the framework because the pc capabilities. When we go over more than 10 thread which consider beyond the saturation point, and no performance advantage resource that called the bottleneck. The bottleneck point that point we cannot increase parallel program performance and scalability but also reduce the work. In our situation we found our bottleneck at thread 11 for that reason we got our thread parameters form thread one to ten.

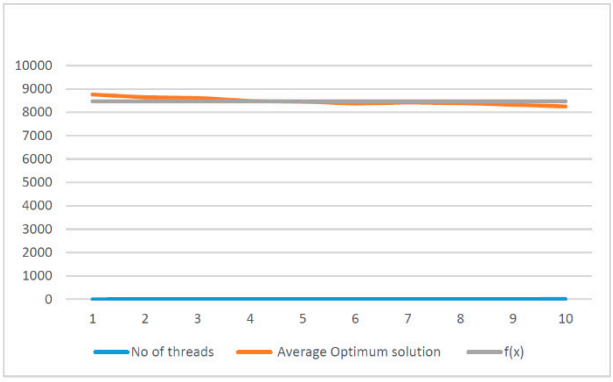
The table 3, we observe that the linear polynomial $\mathbf{F}(\boldsymbol{x}) = \mathbf{8470} - \mathbf{0.205}\,\boldsymbol{X}$ which gives slightly shorter time where x = [1, 2, 3, 4]. Then when we lunch more threads it obviously that framework slightly performed better than f(x). But if we make x=200, we will get better feasible solution 8429 and we are not able to lunch 200 threads within our framework.

No of Threads	$\mathbf{F}(\boldsymbol{x}) = \mathbf{8470} - \mathbf{0.205}\,\boldsymbol{X}$	Average of Optimum
1	8,469.80	8759.75
2	8,469.59	8645.18
3	8,469.38	8618.24
4	8,469.18	8485.11
5	8,468.97	8448.81
6	8,468.77	8378.67
7	8,468.56	8418.53
8	8,468.36	8390.70
9	8,468.15	8320.90
10	8,467.95	8247.50

[4] Comparison between the optimum solution from f(x) and framework

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 November 2016 doi:10.20944/preprints201611.0033.v1

However, the best optimum solution for berlin52 is 7542, and we can obtain this solution by making x= 4,965. Furthermore, better result produced such as 7445 when using x= 5000. Conversely, there is no guarantee to get close optimum solution by using ordinary GA algorithms. Instead, we can get sort of solution using linear f(x). Besides, it provide verification of the solution method if it is good or not. Again, if we compare column 2 with column 3, easily we can verify the quality of optimum solutions.



(**Figure 2 average optimum solution and model f(x)**)

In figure 2, we use a line chart to show over all comparison of the 10 threads and solutions. In the chart above, both the result are almost overlapping. Which means there is no such difference in the output of both the results. There is a strong positive correlation between average optimum solution and Model f(x). That means with the increase of Model f(x), the average optimum solution value should be increase and vice versa.

7. Conclusions

TSP is classified as NP completeness problem. The main purpose of this study to investigate the correlation between the number of cores and optimum solutions. This correlation presented in linear polynomial equation.

We observed the results from our framework and infer the equation that used regression model for demonstrate this correlation in linear polynomial equation. This polynomial equation gives ability to better predicate list of optimum solutions for this specific instance TSP problem.in additional better predicate of the impact of thread and expect performance of thread allocation.

Furthermore, this model shows that value of feasible optimum solution is function of value of number of GAs that mean the value of solution always dependent on value of number of GAs. It is very good tooling to analysis the relation between genetic algorithm parameters. Also, f(x) could use to evaluate GAs are perform.

Yet, this technique using the linear F (x) allow to transfer problem of TSP from NP-completeness class to NP problem class where we able to verify the optimum solutions. Moreover, the best results of TSP were produced without local search.

We can use this f(x) function to lead to better solutions and for predicating in advance the list of possible solutions. In fact, those solutions can bound the exponential algorithms to be provably

efficient. The bounded property that makes polynomial algorithms are preferred way for solving NP problems. Moreover, find the thread allocation that guarantees certain response.

From all of the charts we used for this experiment is clearly shown that the function f(x) developed has very close results with the framework results. In addition, the f(x) can find global optimum solution, but the framework there is no guarantee to find the global optimum.

.

Author Contributions:

We are in this study, did experiment to discovery the mathematical correlation between the number of threads and optimum solution in parallel genetic algorithm. This is new approach in parallel processing research that consider the number of threads as parameter that has crucial impact. We did analysis of optimum results and found the linear model that represent the relation and we had approved that model can use to predicate impact of threads and it can use to estimate the expect performance for any thread allocation.

References

[1] M. R. Gary and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness," ed: WH Freeman and Company, New York, 1979.

[2] E. Bagheri and H. Deldari, "Dejong function optimization by means of a parallel approach to fuzzified genetic algorithm," in *11th IEEE Symposium on Computers and Communications (ISCC'06)*, 2006, pp. 675-680.

[3] R. Bardenet, "Towards adaptive learning and inference-Applications to hyperparameter tuning and astroparticle physics," Université Paris Sud-Paris XI, 2012.

[4] E. Cantu-Paz, *Efficient and accurate parallel genetic algorithms* vol. 1: Springer Science & Business Media, 2000.

[5] W. A. Chaovalitwongse, C.-A. Chou, T. Y. Berger-Wolf, B. DasGupta, S. Sheikh, M. V. Ashley, *et al.*, "New optimization model and algorithm for sibling reconstruction from genetic markers," *INFORMS Journal on Computing*, vol. 22, pp. 180-194, 2010.

[6] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 1987, pp. 41-49.

[7] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations* vol. 12: Springer Science & Business Media, 2006.

[8] J. E. A. Heris and M. A. Oskoei, "Modified genetic algorithm for solving n-queens problem," in *Intelligent Systems (ICIS), 2014 Iranian Conference on*, 2014, pp. 1-5.

[9] M. Mansour, "A Genetic Algorithm identification technique for the estimation of process derivatives and model parameters in on-line optimization," in *Systems and Control (ICSC), 2016 5th International Conference on*, 2016, pp. 120-125.

[10] J. McCall, "Genetic algorithms for modelling and optimisation," *Journal of Computational and Applied Mathematics*, vol. 184, pp. 205-222, 2005.

[11] F. Oppacher and M. Wineberg, "The shifting balance genetic algorithm: Improving the GA in a dynamic environment," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, 1999, pp. 504-510.

[12] M. Lybanon and K. Messa, "Genetic algorithm model fitting," *Practical handbook of genetic algorithms*, vol. 3, 1999.

[13] D. Gupta and S. Ghafir, "An overview of methods maintaining diversity in genetic algorithms," *International journal of emerging technology and advanced engineering*, vol. 2, pp. 56-60, 2012.

[14] P. Guo, X. Wang, and Y. Han, "The enhanced genetic algorithms for the optimization design," in *2010 3rd International Conference on Biomedical Engineering and Informatics*, 2010, pp. 2990-2994.

[15] E. Popovici and K. De Jong, "Understanding EA dynamics via population fitness distributions," in *Genetic and Evolutionary Computation Conference*, 2003, pp. 1604-1605.

[16] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, pp. 17-26, 1994.

[17] S. Paterlini and T. Minerva, "Regression model selection using genetic algorithms," in *Proceedings of the 11th WSEAS international conference on nural networks and 11th WSEAS international conference on evolutionary computing and 11th WSEAS international conference on Fuzzy systems*, 2010, pp. 19-27.

[18] M. Tang and R. Y. Lau, "A parallel genetic algorithm for floorplan area optimization," in *Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)*, 2007, pp. 801-806.

[19] C. Darwin, "On the origins of species by means of natural selection," *London: Murray*, vol. 247, 1859.

[20] S. Wright, "Isolation by distance," *Genetics*, vol. 28, p. 114, 1943.

[21] E. Alba and J. M. Troya, "Improving flexibility and efficiency by adding parallelism to genetic algorithms," *Statistics and Computing*, vol. 12, pp. 91-114, 2002.

[22] E. Alba and B. Dorronsoro, *Cellular genetic algorithms* vol. 42: Springer Science & Business Media, 2009.

[23] G. Luque and E. Alba, *Parallel genetic algorithms: theory and real world applications* vol. 367: Springer, 2011.

[24] A. Kaznatcheev, "Complexity of evolutionary equilibria in static fitness landscapes," *arXiv preprint arXiv:1308.5094*, 2013.

[25] W. LIPENG, "Large-scale structural identification by multi-civilization genetic algorithm approach," 2004.

[26] J. Lis, "Parallel genetic algorithm with the dynamic control parameter," in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, 1996, pp. 324-329.

[27] A. M. Sutton and F. Neumann, "Parameterized Runtime Analyses of Evolutionary Algorithms for the Euclidean Traveling Salesperson Problem," *arXiv preprint arXiv:1207.0578*, 2012.

[28] K. Deb and S. Agrawal, "Understanding interactions among genetic algorithm parameters," *Foundations of Genetic Algorithms*, vol. 5, pp. 265-286, 1999.

[29] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on evolutionary computation*, vol. 3, pp. 124-141, 1999.

[30] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, pp. 95-99, 1988.