Tri-Objective NSGA-II Based Methods for Load Balancing

Siddhartha Dwivedi, Divya Kumar

Abstract—The rapid rise of virtual machines are affecting the daily lives of people profusely. It is clear that to cater to such huge amounts of requests, servers which can withstand the upper bound of those requests must be maintained. In this paper, we propose a model based on Evolutionary Algorithms which attempts to schedule given tasks to virtual machines in such a manner, so as to minimise the load imbalance among the different machines available. We show that using a greedy approach with certain optimisation functions, a workable solution can be reached which would help reduce this "upper bound" mentioned above. Through it, one can expect the load on any particular machine to not exceed a certain amount and be distributed amongst all virtual machines.

Keywords - Genetic Algorithms, Load Balancing, Makespan, NSGA-II, Virtual Machines

I. INTRODUCTION

Server farms today have become huge and are still ever expanding. These server farms have vast costs associated with them, in the form of maintenance. These include warehouse costs, cooling costs and electricity overheads just to name a few. These not only use huge amounts of resources, but also impact the environment negatively. The whole world relies on these servers to meet their routine/excess data demands. These servers are thus designed in a manner to sustain even the largest of tasks assigned to them. This sustenance of the upper limit accompanies with itself even larger resource utilizations mentioned earlier. This paper presents a model which manages to efficiently balance the loads on a number of machines which are available for the tasks. In this, we consider three aspects, using which we aim to minimise load and equally distribute them on the available machines. They were chosen from many functions which give a metric of balanced load, from the survey in [1]. The functions chosen are-

1) Standard Deviation of load on machines.
2) Makespan of schedules on the machines.
3) Linear equilibrium entropy of loads. This is an NP-Hard problem [2]. There is no algorithm which, in polynomial time can guarantee the best solution. However, there is a proven track record of Evolutionary Algorithms in finding a sufficiently good solution, refer to [3] and [4]. In this paper also, we have tried to use the already established principles of Evolutionary Algorithms to Tackle the problem of Optimal Load Balancing. Hence, we present another approach. The proposed model is making use of *Non-Dominated Sorting Genetic Algorithm - II* as a fitness selection model for multi-objective optimisation. We create random solutions and use them as inputs to the above mentioned fitness functions. We present the hypothesis that as we move through the generations of the solutions(each having best traits of the previous generation), the model will converge to an optimum, where the first two functions will be minimised and the last shall be maximised. As a result of the latter, the load on all machines would be balanced, and would reduce the chance of overload greatly. To prove that the model works, we run it on a standard data-set given in [5] for heterogeneous machines. The data-set has different conditions for values of resource utilization. The model will attempt to optimise the functions in all those cases. For more resources and work done in this field, one can refer to [6] and [7]

II. OPTIMISATION PARAMETERS

A. Standard deviation of loads on machines.

The standard deviation of loads is defined as the average deviation from the mean of the loads on the different machines. Mathematically:

For *m* tasks, each being assigned to one of the *n* machines.

The final loads on the machines being-

$\{X_1,X_2,...,X_n\}$

The Standard deviation is given as -

$$dev = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - X_{mean})^2} \quad (1)$$

This value of (1) is to be minimised in subsequent iterations for balanced load among machines.

B. Makespan of loads on machines.

The Makespan of a particular assignment of loads on machines is the largest load which is handled by the given *n* machines.

If in a given assignment of *m* tasks to *n* machines, the final loads are -

$\{X_1,X_2,...,X_n\}$

then,

$$Makespan = \max_{i=1}^{n}(X_i) \quad (2)$$

For efficient load balancing, this value of (2) must be minimised in the iterations of the evolutionary algorithm.

Revised Manuscript Received on December 05, 2019.

*Correspondence Author

Siddhartha Dwivedi*, Department of Information Technology, Motilal Nehru National Institute of Technology, Prayagraj, India.
E-mail: siddharthadwivedi@mnnit.ac.in

Divya Kumar, Assistant Professor, Department of Computer Science and Engineering, Motilal Nehru National Institute of Technology, Prayagraj, India. E-mail: divyak@mnnit.ac.in

C. Linear equilibrium entropy

This is one of the most important metrics as entropy measures the amount of purity or impurity in data. Impurity here refers to data being homogeneously distributed among various classes, and purity refers to data being clustered in one or few classes. We need our load to be distributed evenly among all machines and hence, we must maximise the *Impurity* of the loads. It is given by,

$$Entropy = -\sum_{i=1}^{n} p_i \log p_i \tag{3}$$

where, $p_i = \frac{X_i}{\sum_{k=1}^{n} X_k}$

As above, X_i is the load on the i^{th} machine. The equation for the above parameter was taken from [8]. For more information on standard deviation, makespan and entropy, we redirect the reader to [1].

III. PROPOSED MODEL

In the proposed model, we put forward the use of Evolutionary algorithms. For m objective functions, and n input size, according to [9], the complexity is $O(mn^2)$. Here we have 3 objective functions, and various different values of input data. In the data-set, we have 512 tasks which have different values of time taken as resource utilization on 16 machines. Each task, can and has to be assigned to one of the machines, but not more. The assignment may or may not have a unique utilization value for a particular machine. Thus, many possibilities are present for assignment, in total - $(16)^{512}$. Going through all solutions using NSGA2 guarantees an optimal solution. For a better understanding of the working of NSGA-2, one can refer to [9]. The model follows the given steps

1) Initialisation:

An initial random assignment is made. This assignment is made in the form of a list of size 512. The values of the list vary from $1 - 16$. The i^{th} value in the list refers to the number of the machine on which the i^{th} task is assigned. For example, a value of 12 at the 20^{th} position of the list means that the 20^{th} task is assigned to the 12^{th} machine. In this manner, we initialise a population of 1000 random solutions(this number may be varied). This is our initial random population on which the model will be run.

2) Selection:

From these 1000 initialised solutions, which one gives us the optimised values of all the functions? This is done through NSGA-2. Thus, we sort the 1000 solutions according to NonDominated sorting. Since NSGA-2 always minimises, the entropies are made negative so as to eventually maximise it. Here, we can select the best solutions. The topmost solution gives us the optimised values for all the functions.

3) Crossover/mutation:

These operations are used to explore the variety of solutions in the whole search space. Since, we are using Genetic algorithms, there must be chromosomes on which it works.

These are discussed further in greater detail. For crossover, we keep a probability of 1, so as to maximise the chances of finding an optimal solution. This means that, for n initial solutions, we will create n more children using those parents. Thus, in total we will have a population of $2n$ solutions. Here, we use one point crossover. As an example, consider these two solutions(hypothetical, not actual chromosomes used) as an illustration-

$P_1 = \{1,2,3,4\}$ and $P_2 = \{5,6,7,8\}$

After crossover, we have,

$C_1 = \{1,2,7,8\}$ and $C_2 = \{5,6,3,4\}$ where, P_1,P_2 are parents and C_1,C_2 are children. This helps us to explore the sample space.

After crossover, we perform the mutation operation on the $2n$ solution population. For mutation, we consider a probability of 0.5. Using a random generator, if the generated number is greater that 0.5, we go for mutation, otherwise not. In the mutation function, since we want to retain the best solutions, and not change them, we perform mutation only on the lower solutions which were sorted according to NSGA-2. In this manner, we retain the best solutions and explore more solutions in subsequent iterations. In this model, mutation is performed as follows-

We take the bottom 100 solutions and randomly change values of these solutions. We generate 50 random indices which denote one of the 100 bottom-most solutions, for each such solution, we randomly select 10 tasks and change their assignments. This also helps us in exploring different solutions, and in retaining the best solutions. As an illustration, consider chromosome- 0

$C = \{1,2,3,4\}$ is mutated to $C = \{1,7,3,4\}$

The chromosomes used in the model are discrete and realvalued. The length of a single chromosome is 512 spaces. Each space represents the task which must be allocated. These spaces are filled by values ranging from $1 - 16$, each value representing the machine to which that particular task was allocated. The crossover and mutation operations are performed on these chromosomes. Many different chromosomes together make an input population. Consider for example,

A single chromosome[1,3,5,12,...,14,13,2,16,5,8]

An array of chromosomes[1,3,5,12,...,14,13,2,16,5,8], [5,4,3,2,...,12,11,9,7,3,1],.upto n(*input*)

This array of Chromosomes forms the input to our model. There may be many ways to choose a chromosome, for example, Binary Form. Although, it seems like a good choice, it would only increase the dimensions of the input which is to be given to the model. This is because each number representing a machine would be written in binary form. Hence, we have stuck to the Real-valued representation of the chromosome, changing the mutation and crossover operators accordingly, as mentioned above.

This forms the basic skeleton of the proposed model. After the $2n$ population has undergone mutation, we select the topmost n solutions to be parent for the next iteration of the algorithm. We run this algorithm for 100 generations and gather the optimal solutions in each generation. The results for the data-set have been summarized. A flowchart of the model is shown in Fig. 1

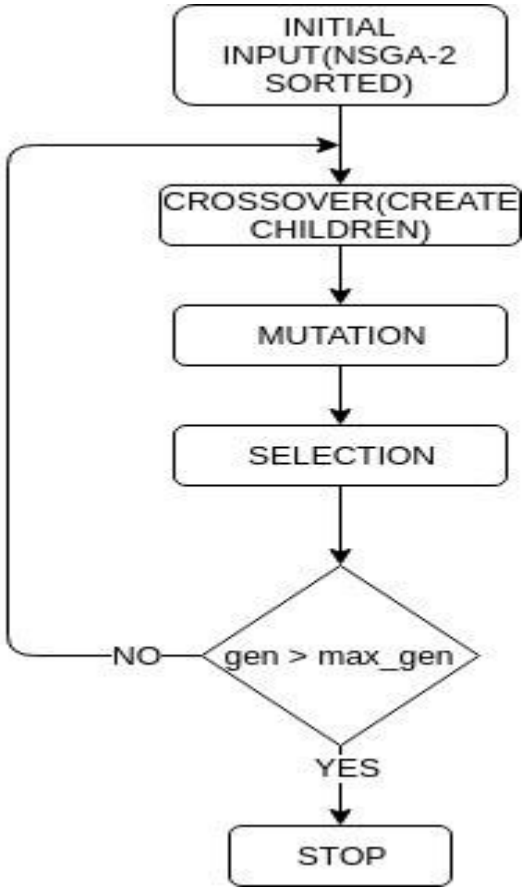

Fig. 1: PROPOSED MODEL

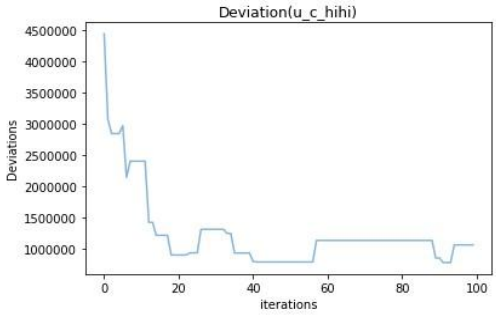

Fig. 2: Deviations for u_c_hihi

IV. RESULTS AND CONCLUSIONS

The model was run on the benchmark [5] heterogeneous data-set. The results are summarized in the Algorithm 1

NSGA-2 based load balancing *population* ← *n gen* ← 1
inputs ← *CreateInputs*(*n*) *FunctionV alues* ← *calcV alues*(*inputs*) *SortedInputs* ← *NSGA*(*FunctionV alues*) while *gen* ≤ *max_gen* do *children* ← *crossover*(*SortedInputs*) *NewSolutions* ← *SortedInputs* + *children*
MutatedSolutions ← *mutation*(*NewSolutions*)
NewV alues ← *calcV alues*(*MutatedSolutions*)
SortedNew ← *NSGA*(*NewV alues*) *NextGen* ← *EmptyList*()
while *sizeof*(*NextGen*) 6= *n* do if *sizeof*(*NextGen* + *front*) ≤ *n* then *NextGen.add*(*front*) else
while *sizeof*(*NextGen*) 6= *n* do *NextGen.add*($front_j$) end while end if end while
gen ++
end while

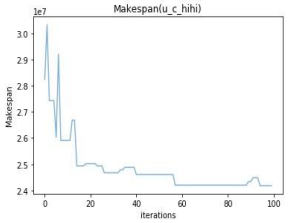

Fig. 3: Makespans for u_c_hihi

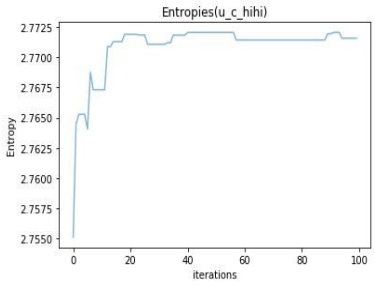

Fig. 4: Entropies for u_c_hihi

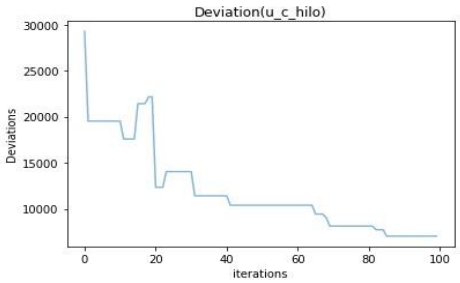

Fig. 5: Deviations for u_c_hilo

Graphs above. It is clear that for all the different types

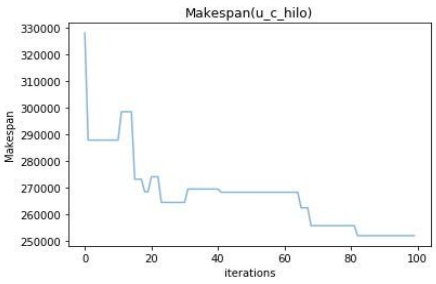

Fig. 6: Makespans for u_c_hilo

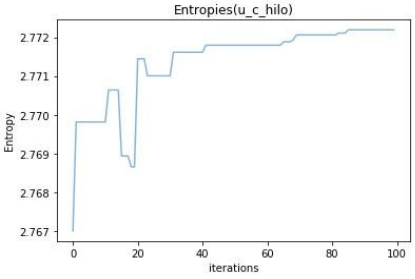

Fig. 7: Entropies for u_c_hilo

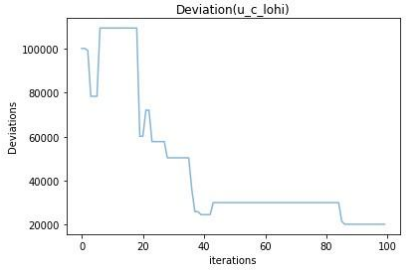

Fig. 8: Deviations for u_c_lohi

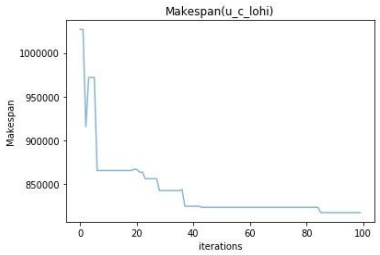

Fig. 9: Makespans for u_c_lohi

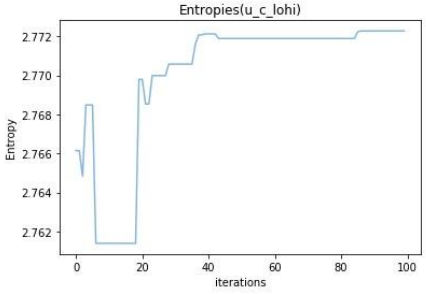

Fig. 10: Entropies for u_c_lohi

(*consistent*,*inconsistent*,*semi − consistent*(*c*,*i*,*s*)) and their corresponding subtypes (*hihi*,*hilo*,*lohi*,*lolo*), the functions converge towards an optimum in subsequent iterations. The simulation was run on an Intel *Core − i*5 processor, with 8 GB RAM. The results may be compared with [10]. Some noteworthy points and inferences from the graphs are-

1)The model runs for 100 iterations. It was first ran for 20 as well as 50 iterations. For the previous two cases(20,50),

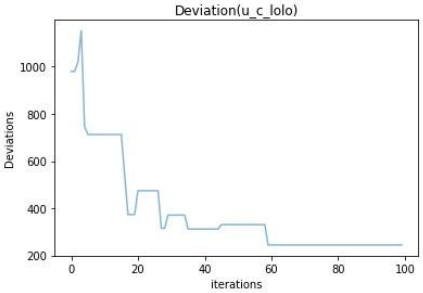

Fig. 11: Deviations for u_c_lolo

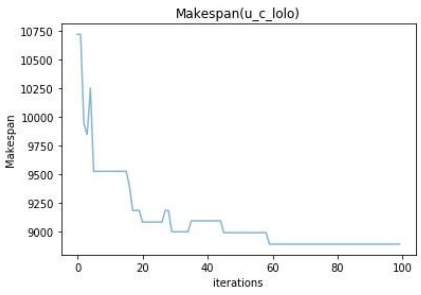

Fig. 12: Makespans for u_c_lolo

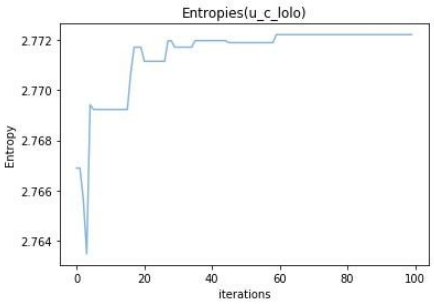

Fig. 13: Entropies for u_c_lolo

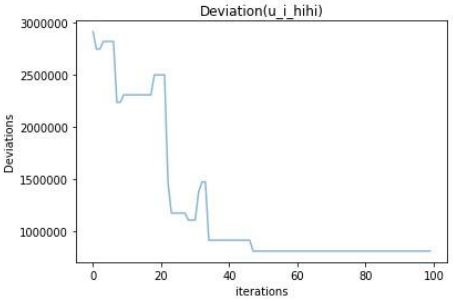

Fig. 14: Deviations for u_i_hihi

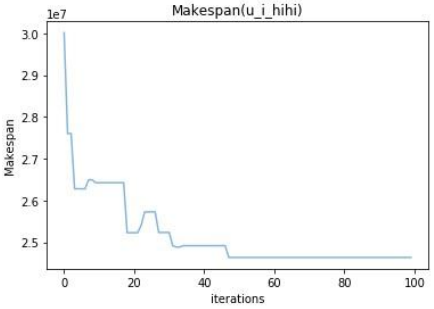

Fig. 15: Makespans for u_i_hihi

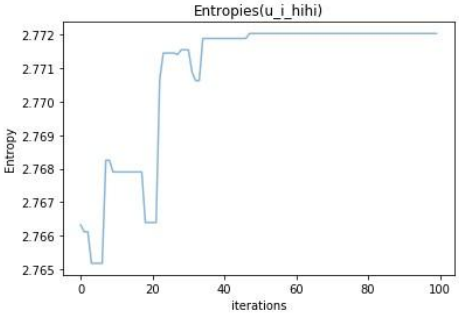

Fig. 16: Entropies for u_i_hihi

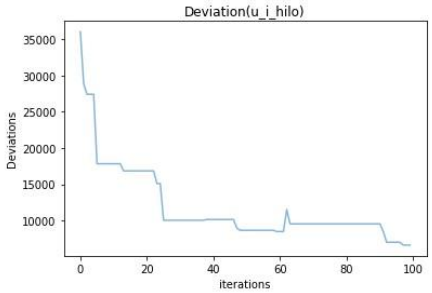

Fig. 17: Deviations for u_i_hilo

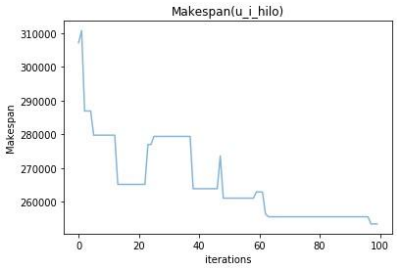

Fig. 18: Makespans for u_i_hilo

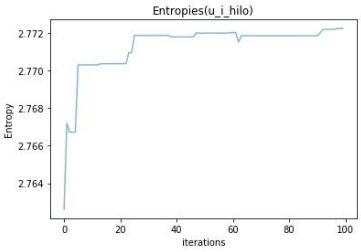

Fig. 19: Entropies for u_i_hilo

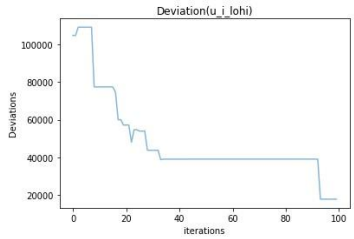

Fig. 20: Deviations for u_i_lohi

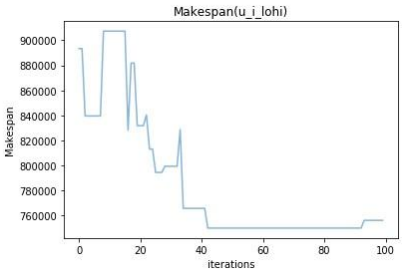

Fig. 21: Makespans for u_i_lohi

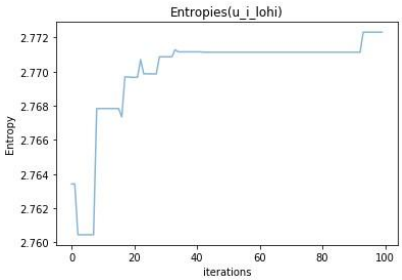

Fig. 22: Entropies for u_i_lohi

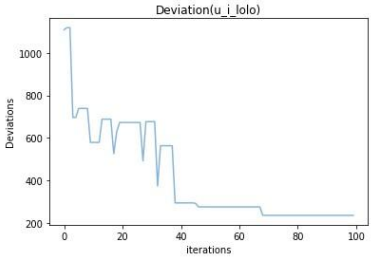

Fig. 23: Deviations for u_i_lolo

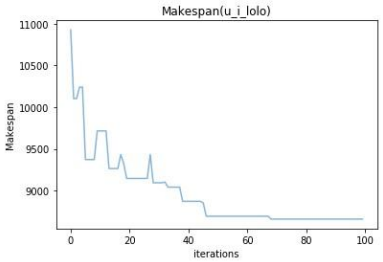

Fig. 24: Makespans for u_i_lolo

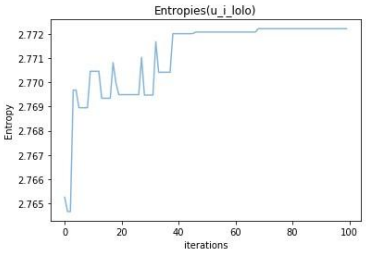

Fig. 25: Entropies for u_i_lolo

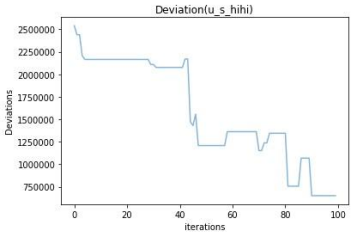

Fig. 26: Deviations for u_s_hihi

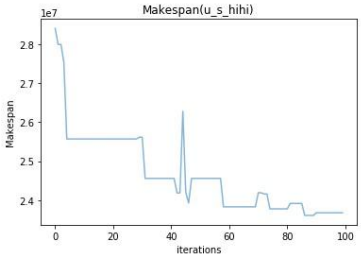

Fig. 27: Makespans for u_s_hihi

the algorithm showed signs of converging, but 100 iterations give conclusive signs for the same. Hence, 100 iterations were chosen. Of course, more the iterations, better would be the solutions. That is, the optimum solution is guaranteed if the algorithm is run for a sufficiently large number of iterations.

2)The labelling of the graph is done as follows-

The x-axis represents the iterations(1 − 100) and the y-axis represents the optimisation parameter values for the particu-

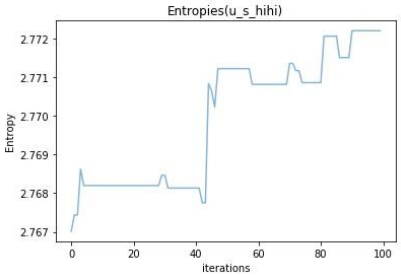

Fig. 28: Entropies for u_s_hihi

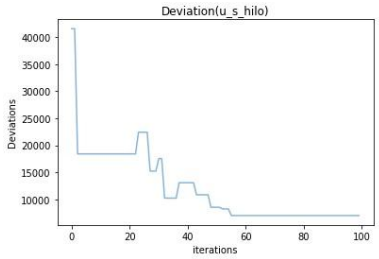

Fig. 29: Deviations for u_s_hilo

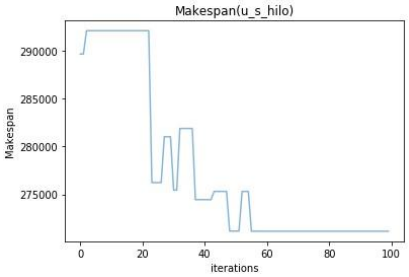

Fig. 30: Makespans for u_s_hilo

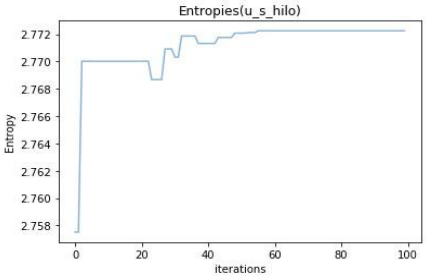

Fig. 31: Entropies for u_s_hilo

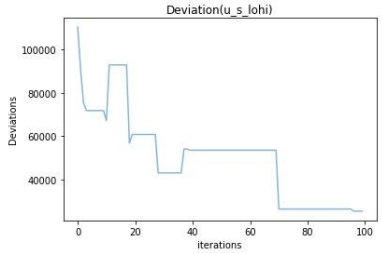

Fig. 32: Deviations for u_s_lohi

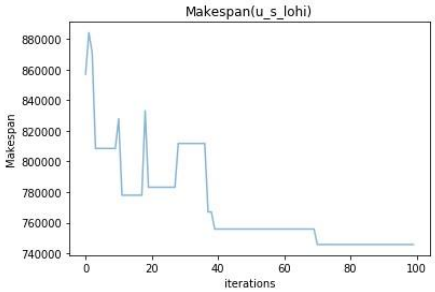

Fig. 33: Makespans for u_s_lohi

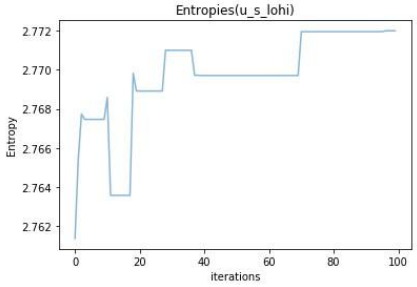

Fig. 34: Entropies for u_s_lohi

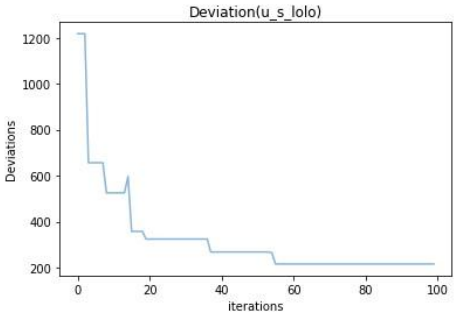

Fig. 35: Deviations for u_s_lolo

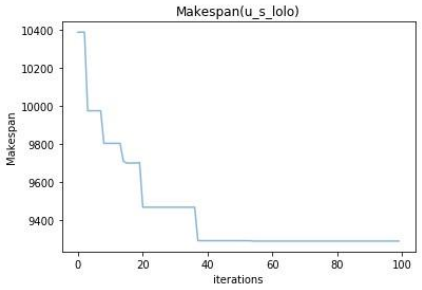

Fig. 36: Makespans for u_s_lolo

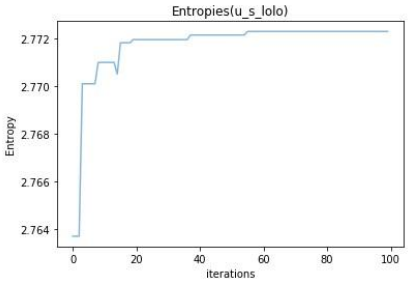

Fig. 37: Entropies for u_s_lolo

lar dataset, i.e either *Deviations*,*Makespan* or *Entropy*. The title of the graph represents the dataset type in brackets. 3)The final values are always more optimised than in the initial iterations(lesser for first two functions and more for the third function).

4)In some cases, the graph may move towards a non-optimal value. This is always accompanied by a simultaneous increase(or decrease) to a better value in one or more of the other functions. This happens due to the working of the NSGA-2. That particular optimum solution might decrease the value of one function, but overall, it is the best solution amongst all those present in that particular population of a generation and provides the most balanced loads. 5)The graphs convey that the model will converge to the best solution eventually.

6)Since, this assignment is an NP-Hard problem, it is not possible to search the whole space efficiently. It was discussed above that the total number of solutions are $(16)^{512}$. Working on only 1000 solutions of them and getting balanced loads from only them would seem a very lucrative approach.

7)In all the cases, the decrease in standard deviation is at-least 50%. This gives us highly balanced loads on the machines, thus achieving the hypothesised results.

FURTHER WORK

This model can be further improved using other functions from the survey [1] along with the ones already used. Also, this model gives results for heterogeneous data, i.e. independent data-set. We can do this for tasks which are dependent on each other. We could prove that this algorithm converges even for this latter case.

REFERENCES

1. M. Xu, W. Tian, R. Buyya, "A Survey on Load Balancing Algorithms for Virtual Machines Placement in Cloud Computing", CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE,
2. Wiley InterScience, Volume 29, Issue 12, February 2017
3. A. Gamal, Y. Hamam. "Two phase algorithm for load balancing in heterogeneous distributed systems." 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2004. Proceedings.. IEEE, 2004.
4. D. Kumar, K.K. Mishra. "Multi-Objective Optimization using CoVariance Guided Artificial Bee Colony." Journal of Information Science Engineering 34.2 (2018).
5. D. Kumar, K.K. Mishra. "Incorporating logic in artificial bee colony (abc) algorithm to solve first order logic problems: The logical abc." 2015 7th International Conference on Knowledge and Smart Technology (KST). IEEE, 2015.
6. F.N. Braun, https://code.google.com/p/hcsp-chc/source/browse/trunk/AE/ProblemInstances/HCSP/Braunetal/uchihi.0?r=93. Accessed on September 2019
7. R. Wang, W. Le, X. Zhang, "Design and Implementation of an Efficient Load-Balancing Method for Virtual Machine Cluster Based on Cloud Service".
8. K. Dasgupta, B. Mandal, P. Dutta, J. K. Mondol, S. Dam, "A Genetic Algorithm(GA) based Load Balancing Strategy for Cloud Computing", International Conference on Computational Intelligence: Modeling Techniques and Applications(CIMTA) 2013.
9. C.Wang, Z. Zhou, X. Mao, S. Lin, "A Quadratic Equilibrium Entropy Based Virtual Machine Load Balance Evaluation Algorithm", International Conference on Intelligent Systems Research and Mechatronics Engineering (ISRME 2015)
10. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 6, NO. 2, APRIL 2002
11. S. K. Panda, P. K. Jana, "An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems", Springer Science+Business Media, LLC, 30 October 2018.

AUTHORS PROFILE

Siddhartha Dwivedi, holds Bachelor of Technology in Information Technology at Motilal Nehru National Institute of Technology, Allahabad. He completed his Intermediate with a percentage of 95.4 from
Delhi Public School, Kalyanpur (affiliated to CBSE). He has pursued research in Load Balancing and Scheduling problems through Genetic Algorithms.

Divya Kumar. is an Assistant Professor at Motilal Nehru National Institute of Technology. He has completed is PhD and M.tech(Gold Medalist) from the same. Before this, he had completed his Bachelor of technology degree from U.P Technical University(Gold medalist). He has also held the position of Senior Software Developer at NetApp India Ltd for more than a year. His research interests include but are not limited to Genetic Algorithms and Multi-objective Optimization. The various publications can be found on his CV on the Institute website: *www.mnnit.ac.in*