A Parallel Genetic Algorithm for Solving Deadlock Problem within Multi-Unit Resources Systems

Rabie Ahmed[1, 2, *], Taoufik Saidani[1], and Malek Rababa[1]
[1] Department of Computer Science, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia
[2] Department of Mathematics and Computer Science, Faculty of Science, Beni-Suef University, Beni-Suef 65211, Egypt
[*] Correspondence Author:

Summary
Deadlock is a situation in which two or more processes competing for resources are waiting for the others to finish, and neither ever does. There are two different forms of systems, multi-unit and single-unit resource systems. The difference is the number of instances (or units) of each type of resource. Deadlock problem can be modeled as a constrained combinatorial problem that seeks to find a possible scheduling for the processes through which the system can avoid entering a deadlock state.

To solve deadlock problem, several algorithms and techniques have been introduced, but the use of metaheuristics is one of the powerful methods to solve it. Genetic algorithms have been effective in solving many optimization issues, including deadlock Problem. In this paper, an improved parallel framework of the genetic algorithm is introduced and adapted effectively and efficiently to deadlock problem. The proposed modified method is implemented in java and tested on a specific dataset. The experiment shows that proposed approach can produce optimal solutions in terms of burst time and the number of feasible solutions in each advanced generation. Further, the proposed approach enables all types of crossovers to work with high performance.

Key words:
Deadlock Multi-instances, Genetic Algorithm, Banker Algorithm, Parallel Algorithms.

1. Introduction

A deadlock in an operating system happens when a process or thread enters a waiting state because a resource sought by it is held by another waiting process, which is waiting for yet another resource. A system is said to be in a deadlock if a process is unable to modify its state indefinitely because the resources sought by it are being utilized by another waiting process [1]. Deadlock occurs when software and hardware locks are employed to manage shared resources and accomplish process synchronization in multiprocessing systems, parallel computing, and distributed systems [2].

If all of the following circumstances exist in a system at the same time, it can result in a deadlock:

1. Mutual Exclusion: There must be at least one non-shareable resource [1]. At any given moment, only one process can use the resource.
2. Hold and Wait or Resource Holding: A process presently has at least one resource and is requesting more resources from other processes.
3. No Preemption: The operating system must not de-allocate resources that have already been assigned; instead, the holding process must release them willingly.
4. Circular Wait: A process must be waiting for a resource that is being held by another process, which is in turn waiting for the resource to be released by the first process. P1, P2, ..., PN are a collection of waiting processes in which P1 is waiting for a resource owned by P2, P2 is waiting for a resource held by P3, and so on until PN is waiting for a resource held by P1 [1],[3].

From Edward Coffman's original explanation in a 1971 article, these four conditions are known as the Coffman conditions. A deadlock can't happen if any of these requirements aren't met [3].

Multi-unit and single-unit resource systems are the two types of systems available. Each sort of resource has a different number of instances (or units). In a single-unit resource system, each resource has just one instance to distribute to many processes. If a cycle forms in the related Resource Allocation Graph (RAG) in a single-unit resource system, the system is deadlocked. Multi-unit resource systems, in which any number of instances of a particular resource type can exist, are more difficult. A cycle in the related RAG for a multi-unit system, unlike single-unit RAGs, provides no information about system deadlock. As a result, a multi-unit resource system is the most generic form of a single-unit resource system. Consequence, a multi-unit deadlock detection technique may be utilized for single-unit resource systems, but not the other way around.

Most modern operating systems are incapable of preventing a deadlock [1]. When a deadlock occurs, various operating systems react in a variety of non-standard ways. The majority of techniques operate by inhibiting one of the four Coffman requirements, often the fourth [4]. The following are the major approaches:

Manuscript received December 5, 2021
Manuscript revised December 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.12.25

1. **Deadlock Prevention:** By making sure that at least one of the deadlock requirements is broken.
2. **Deadlock Avoidance:** By giving a priori data, the system will be able to forecast and avoid deadlock situations.
3. **Deadlock Detection:** Detecting and resolving deadlock situations.

Deadlock prevention limits concurrency, but deadlock recovery can be time-consuming and costly. Because of its considerable cost, deadlock avoidance provides complete concurrency but is rarely employed. The bankers' algorithm was created by Dijkstra [5], and Haberman [6] improved it to include numerous resource types. It supports maximum concurrency in a system with n processes and m various resource kinds, and its run time is O(m x n). There are two definitions are needed to discuss the algorithms:
DEFINITION 1: A safe sequence of processes (P1, P2,..., Pn) is considered a safe state if each Pi's remaining resource requirements can be met with the available resources plus the resources kept by all the Pj with j < i.
DEFINITION 2: The system is in a safe state if the processes in the system follow a safe sequence. The system's state is unsafe if no such sequence happens.

If the resources required by process Pi are not immediately accessible, Pi will wait until all of Pj has completed, where j< i. Pi will then get all of the resources it needs, do the work it was given, return all of the resources it was given, and exit. Pi+1 can acquire its needed resources once Pi has completed, and so on. According to Haberman [6] where each process's maximum resource need (claim) is known ahead of time, this study assumes that each process may make resource requests in any order within this constraint. Depending on the notion of a safe state, the banker's algorithm ensures that the system will never deadlock; in other words, the concept effectively ensures that the system will remain in a safe state. Initially, the system is in a stable and secure condition. When a process requires a resource that is already available, the system must determine whether the resource may be allocated immediately or if the process must wait. Only if the request leaves the system in a safe condition, that is, if a safe sequence of operations occurs within the system, should it be granted.

The suggested parallel approach in this research is a two-phase algorithm in which a hybrid of genetic algorithms and banker's algorithm was devised and developed in two stages, the first of which involves extracting features and the second of which involves using a modified parallel Genetic Algorithm.

The rest of the paper is structured as follows. Section 2 examines the linked work. The evolutionary algorithm devised for the deadlock situation is presented in Section 3. The experimental findings are presented in Section 4. In Section 5, certain conclusions are formed.

2. Related Work

This section gives an overview of the strategies presented to solve the deadlock problem, as well as certain techniques that are relevant to our study. In general, the study community focuses on three types of deadlock problems: prevention, avoidance, and detection and recovery, which arise from various problem-solving tactics. Deadlock prevention [7] makes use of system architecture and techniques to prevent the system from being stuck. Deadlock avoidance [8] is event-driven and avoids behaviors that can lead to a deadlock. These two approaches frequently result in resource underutilization. Deadlock detection is a technique for detecting and resolving deadlocks.

The Resource Allocation Graph (RAG) [9] is used to address single-unit resource systems, in which each resource has only one instance of that kind to distribute to different processes. There is a deadlock in this sort of system if a cycle is generated in the corresponding Resource Allocation Graph (RAG). It is more challenging in multi-unit resource systems, because any number of instances of a particular resource type might exist. Unlike single-unit RAGs, the cycle in the corresponding RAG for the multi-unit system provides no information on the system's deadlock. There has been a lot of study on deadlock detection in a variety of fields, but little has been done in the area of optimization strategies. The method of preventing deadlocks in any system design by arranging resources in such a way that at least one of the requisite deadlock conditions is never met. In this field, a large number of research have been presented. The authors presented the reinforcement learning scheduling technique in [10]. This approach is utilized in job-shop discrete production systems and corresponds with high-level deadlock detection. The system was without buffering and the first detection approach proposed to the second level and third level deadlocks. By continuing, the high level deadlock detection algorithm developed in the context of less buffer of the job-shop system using the reinforcement learning scheduling algorithm. In [11], the authors introduced an efficient policy for deadlock avoidance. The heuristic-based parameterized Banker's algorithm is one of the most efficient algorithms available (H-pBA). Due to the first buffer integration; first serve policy in the system, the new algorithm achieves higher outcomes. In [12], [13] a new scheduling algorithm proposed by the authors. The algorithm merges a powerful supervised control with heretical search. The goal of this research is minimizing the make span of part list. Depending on the system reachability, the latest algorithm produces a new heuristic function associated with two rules for dispatching.

A dynamic polynomial window search algorithm was developed by the authors. In [14] the authors concentrate on the automated manufacturing system deadlock control with the failure of multiple resources. The methodology incorporates two steps. The first one is process of siphon without unreliable resources. The solution was achieved through adding optimal deadlock. Then, the unreliable resources controlled by adding new control to guarantee that it can be marked once the resources failed.

A new technique of steady state genetic algorithms combined with the banker's algorithm is introduced in that paper. Extracting features to feed the Genetic algorithm optimizer is the first step in the operation sequence. The structure of the chromosome created in this approach is the one operation processes correlated with three different GA operators: One-point crossover (1X), two-point crossover (2X), and uniform crossover (UX) are the three operators. The introduced method ensures a vast number of near-optimal solutions avoiding the Deadlock system as a safe state.

3. A modified Parallel Genetic Algorithm

The proposed method consists of a two-phase algorithm in which a hybrid of Genetic Algorithms with Banker's Algorithm was designed and developed based on two stages. In the first stage, a features extraction is performed and in the second stage, the proposed modified Genetic Algorithm is applied. In order to employ banker's algorithm to solve Deadlock using Genetic Algorithm, the former requires various inputs to be preprocessed where all these inputs are in the form of arrays named: Max, Allocation, Resources, Need, Free Resources and Burst Time.

The proposed method in this paper is performed on dataset defined by Ahmed NT et al [15] that is generated by employing different operational information representing the general obstacles of deadlock in the real world. The first stage that acts as data preprocessing involves firstly preparing the Max array which is an n x m matrix representing the maximum number of instances of each resource that a process can request. In other words, when Max[i][j] = x, that means the process P(i) can request at most x instances of resource type R(j) as shown in Table (1).

Table 1. Max array

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
P0	15	22	25	10	7	15	10	14	12	19
P1	18	7	20	7	9	8	8	13	16	20
P2	17	10	24	8	8	25	7	12	16	18
P3	14	15	23	8	10	20	15	11	16	20
P4	10	20	20	6	12	21	13	9	13	19
P5	7	19	16	11	10	17	11	8	14	25
P6	8	21	15	10	11	11	12	12	10	20
P7	15	23	7	9	12	15	10	8	11	7
P8	19	5	18	2	13	13	9	7	12	15
P9	9	7	15	5	5	12	10	6	9	14

Secondly, preparing Allocation array that is n x m array representing the number of resources of each type that are currently assigned to each process. So, if Allocation[i][j] = x, that means process P(i) is currently assigned x instances of resource type R(j) as it appears in Table (2).

Table 2. Allocation array

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
P0	0	1	2	2	1	3	1	1	2	1
P1	1	1	4	0	1	2	1	0	1	0
P2	1	1	2	1	0	4	0	2	2	1
P3	0	1	1	1	1	3	0	2	2	2
P4	2	2	4	1	1	0	2	1	2	3
P5	1	2	1	1	1	2	1	0	0	3
P6	4	1	3	0	1	4	0	2	1	0
P7	0	0	1	1	1	3	2	1	1	4
P8	0	0	2	0	0	0	1	0	0	1
P9	4	2	3	1	2	2	1	1	1	2

Thirdly, preparing Resources array that consists of number of instances of all resources in the whole system. So when Available[j] = x, that indicates x instances are available of resource type R(j) as demonstrated in Table (3).

Table 3. Resources array

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
31	39	35	35	26	35	36	33	32	30

Finally, Burst time array is prepared which provides the total time each process requires to implement as shown in Table **(4)**. This array plays a main role in the proposed algorithm, where it is the criterion used to evaluate each solution.

Table 4. Resources array

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9
30	20	25	40	22	50	30	35	10	19

The remaining inputs such as Need array and Free Resources array are derived from the arrays mentioned above. As for Need array that is an n x m array as illustrated in Table (5), where it determines the remaining resource needs of each process. So when Need[i][j] = x, that means process P(i) may need x more instances of resource type R(j) in order to implement its task completely. Calculating the required resources for each process is achieved by applying the Equation (1) as illustrated below:

$$\mathbf{Need_{ij} = Max_{ij} - Allocation_{ij}} \quad \mathbf{(1)}$$

Where: i indicates a process, j indicates a resource.

Table 5. Need array

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
P0	15	21	23	8	6	12	9	13	10	18
P1	17	6	16	7	8	6	7	13	15	20
P2	16	9	22	7	8	21	7	10	14	17
P3	14	14	22	7	9	17	15	9	14	18
P4	8	18	16	5	11	21	11	8	11	16
P5	6	17	15	10	9	15	10	8	14	22
P6	4	20	12	10	10	7	12	10	9	20
P7	15	23	6	8	11	12	8	7	10	3
P8	19	5	16	2	13	13	9	6	12	15
P9	5	5	12	4	3	10	9	5	8	12

Free Resources array representing free instances of all resources in the whole system depicted in Table **(6)**. That can be computed using Equation **(2).**

$$\textbf{Free Resources=Resources} - \sum_{i=0}^{n-1} \boldsymbol{Allocation_{i,j}} \quad \mathbf{(2)}$$

Where:

i: number of process.

j: number of resources starting from 0 to m.

***n*:** total number of processes.

Table 6. Free Resources array

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
18	28	12	27	17	12	28	22	20	14

In the second stage, the proposed modified genetic algorithm is performed, where its inputs are the arrays provided in the first stage. Genetic algorithms are fit to work with a population of possible problem solutions, where each solution called a chromosome also (both terms are used interchangeably in this paper) contains an arrangement of processes in a random order, and is considered safe if that arrangement ensures that all processes are executed.

Each solution provides three pieces of information representing chromosome structure as depicted in the figure (1). The first section of the solution (F1) indicates its state - whether it is a safe solution or not. The second section is the Fitness value (F2), represents how good a solution is. The third section stores a random order of processes waiting to be executed as shown in the third section (F3). The algorithm assigns fitness value for each solution by computing the average waiting time using Fitness Function in the Equation (3). Therefore, solutions with low Fitness values will have a higher probability of being selected for survival than solutions with high fitness values.

$$\text{Average Waiting Time}= \frac{\sum_{i=0}^{n-2}\sum_{j=0}^{i} \boldsymbol{burstTime(P_i)}}{\boldsymbol{n}} \quad (3)$$

Where:

n: the number of processes.

i: a process order in the arrangement

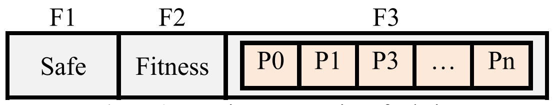

Figure 1. Genetic representation of solution

The proposed modified method as shown in the figure (2) starts by generating the first generation randomly that consists of a number of solutions N, where N is the population size determined in advanced. Algorithm (1) depicts the whole procedure.

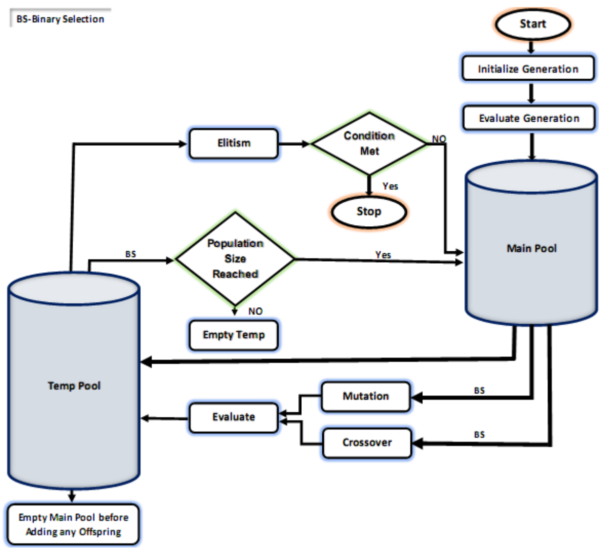

Figure 2. Schematic Diagram of Modified GA

Once the first generation internalized each solution evaluated to compute its fitness value and to determine it is safe or not Algorithm (2) and (3) describe how this evaluation achieved. Then, all solutions are stored in a pool named the Main Pool.

Algorithm 1 Pseudo Code for Adapted Genetic Algorithm

```
1:G ← Generate first Generation
randomly with size N
2:  MainPool← Evaluate(G)
3:  For i = 0 to T, where T is Number of Generations
4:      For j = 0 to N
5:TempPool←Evaluate(Muataion(BinarySelection(MainPool)))
6:TempPool←Evaluate(Crossover(MainPool_j,BinarySelection(MainPool)))
7:      End For
8:      TempPool ←  Add All Solutions of MainPool
9:      Empty MainPool
10:     ElitismList ← Select Best m Solutions from TempPool
11:     MainPool ← ElitismList
12:     For j = 0 to N-m
13:         S ← BinarySelection(TempPool)
14:         MainPool ← S
15:         Remove S from TempPool
16:     End For
17:     Empty TempPool
18:     if termination condition is met then
19:         break;
20:     End if
21: End For
```

Algorithm 2 Pseudo Code for Evaluation

```
1: input: S, where S is a Solution ( Chromosome)
2: Z ← size of S, where Z is the number of Processes
3: Safe ← True
4: For i = 0 to Z
5:     For j =0 to R, where R is the number of resources
6:         if Need_S_ij > Free_j
7:             Safe ←  False
8:             break;
9:             break;
10:        End if
11:    End For
12: End For
13: if Safe = True then
14:    For j = 0 to R
15:        NewFree_j=Allocation_S_ij+ Free_j
16:    End For
17:    Fitness_S ← ComputingWaitingTime(S)
18: esle
19: WorstWaitingTime ← ComputingWaitingTime(S~),
        where the processes is ordered decreasingly by
        their burst time in S~
20:        WorstFitness ←  (Z-D)* WorstWaitingTime,
where D is the number of processes that were able to execute
21: End if
```

Algorithm 3 Pseudo Code for computing waiting Time

```
1: input: S, where S is a Solution ( Chromosome)
2: totalWaitingTime ← 0
3: For i = 0 to N, where N is the number of processes
4:     waitingTime ← 0
5:     For j = 0 to i
6:         waitingTime+=S_i , where S_i is a process in index number i
7:     End For
8:     totalWaitingTime+= waitingTime
9: End For
```

In order to assure adequate evolution pressure, a Binary selection method [16] is employed in which k solutions (parents) are selected randomly and ordered by their fitness values and then the solution with best fitness value (the lowest value) is selected for the crossover and mutation operators. As for the crossover operator, All types of crossover (One point, Two points and Uniform) [17] used in this research work and in term of mutation operator, the convention type [18] in which two genes (processes) are selected randomly and exchange their positions with each other. Next, all solutions stored in the Main Pool are moved to the Temp pool along with the evaluated solutions yielded after applying those operators. To construct the next generation, the best M solutions in the Temp pool as well as solutions selected from the Temp solution by using Tournament Selection algorithm are moved to the Main Pool, where the number of solutions have to be selected from the Temp pool is equal to (N – M). The whole procedure is repeated excepting generating and evaluating the first generation, either until a good-enough solution is found or a fixed count of iteration is reached.

4. Implementation and Results

The proposed algorithm was programmed using Java language and the experiments were conducted on an Intel Core i7-7700 (CPU @ 3.60 GHz with 8 GB RAM) PC running Windows 10 Pro OS and tested on the dataset developed by Ahmed NT et al[16] as mentioned in Section 3, this dataset is created by the use of various operational data representing the general barriers of the real world deadlock. For each algorithm configuration (crossover type), ten executions were made and the parameter settings of the proposed improved algorithm were determined empirically for the deadlock problem as specified in Table (7). In order not to degrade diversity in the population and also to avoid the algorithm from slowing down too much, a low value was chosen for the tournament size. Therefore, the tournament will have a lower selective pressure compared to other tournament types. In which three solutions (chromosomes) are selected randomly from the population and then, out of each couple, the solution with the least cost is selected. The number of elitism solutions was chosen empirically to preserve elitist solutions and avoid the loss of good solutions once they are identified.

Table 7: Parameter Setting for the proposed algorithm

Parameter	**Value**
Generation Number	1000
Population Size	50
Selection Mechanism	Tournament Selection
Tournament Size	3
Number of Elitism Solutions	5
Crossover Type	One Point, Two Points, and Uniform

By observing the results shown in Tables (8 and 9) and the graphs in figures (3 and 4), it is inferred that the proposed framework of the genetic algorithm is capable to find feasible solutions for all crossover types (One Point, Two Points, Uniform) and the performance is quite promising and can reaches to the optimal solutions after few iterations where all the solutions in the population are feasible, in other words, it is capable of finding a large number of optimal solutions that display the system's safe state, which eliminates the Dead Lock case system.

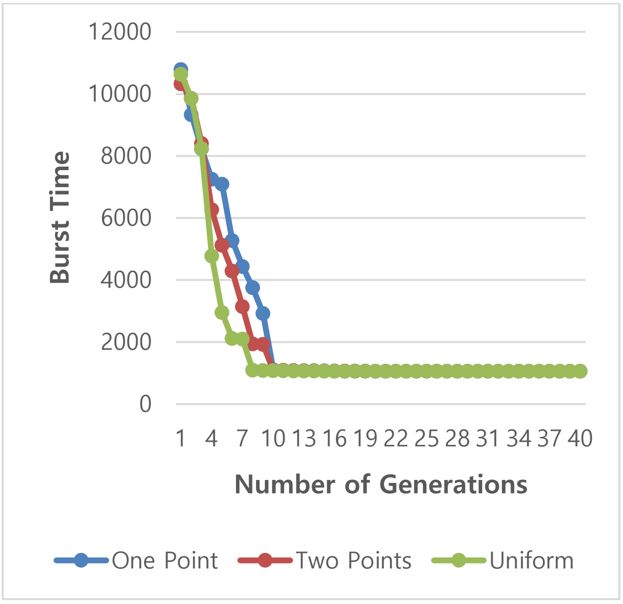

Figure 3. The average burst time of all crossover types for each generation over ten runs

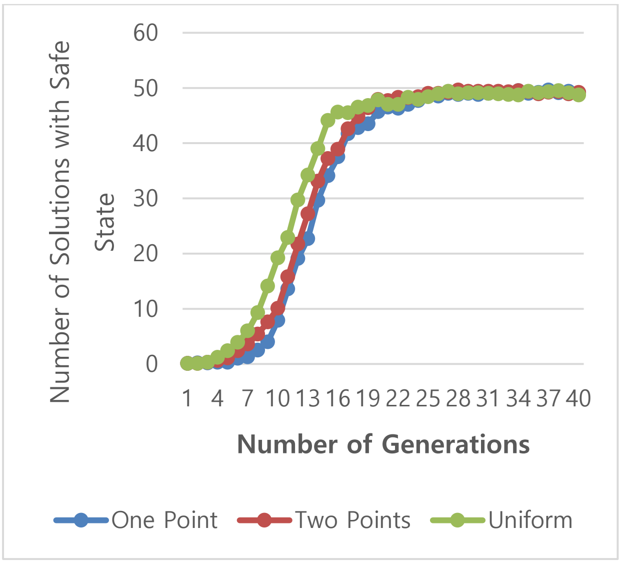

Figure 4. The average number of solutions with safe state of all crossover types for each generation over ten runs

Table 8: Comparison of all crossover types in terms of the required number of generations to reach to the optimal solutions for each run

Run No.	**One Point**	**Two Points**	**Uniform**
1	14	19	14
2	18	22	19
3	22	20	11
4	25	21	19
5	23	16	18
6	13	19	14
7	16	10	19
8	20	17	17
9	16	25	20
10	15	25	17

5.

Table 9: Comparison of all crossover types in terms of the required number of generations to make all the solutions feasible (Safe State) for each run

Run No.	**One Point**	**Two Points**	**Uniform**
1	17	15	15
2	25	19	14
3	17	20	15
4	23	24	16
5	15	17	18
6	27	21	14
7	20	17	16
8	20	17	23
9	16	16	18
10	17	20	19

In fact, all the crossover types lead to high performance, where one point was better than one point and the uniform was the best in terms of the number of the required iterations to reach the optimal solution as shown in figure 5. On the other hand, to reach a population in which all solutions are feasible (Safe State) two points was slightly better than one point and also the uniform surpass the other crossover types as illustrated in figure 6.

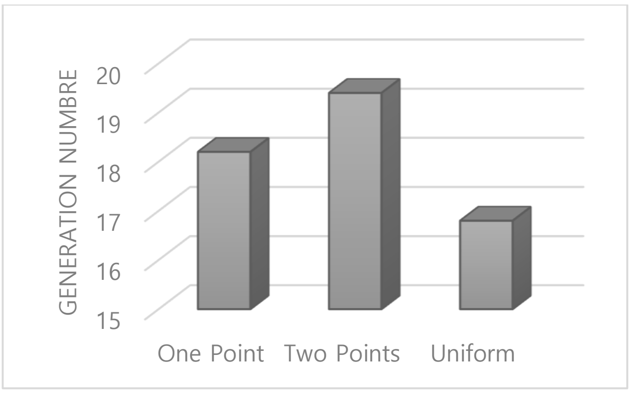

Figure 5. The average required number of generations to reach to the optimal solutions for each crossover type

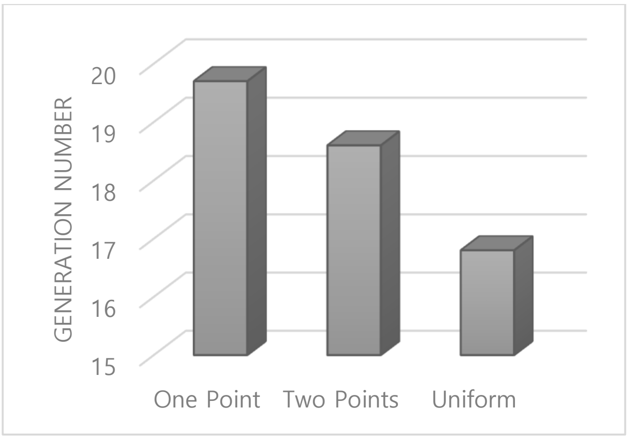

Figure 6. The average required number of generations to make all the solutions feasible (Safe State) for each crossover type

5. Conclusion

In this study, the Hybrid- algorithm has been improved for deadlock problem by proposing an improved framework for the genetic algorithm. Based on the comparisons between the results obtained by the hybrid algorithm and the banker's algorithm prove that the proposed algorithm can reach to the best solution with a reasonable number of generations that means a shorter time and also it is able to produce populations in which all individuals are in safe state with high quality solutions that implies providing many solutions that are able to avoid the deadlock and keeping the system in a safe state. The results also show that the proposed framework enable all crossover types to be utilized in effective and efficient way. In addition, Experimental findings affirm the standard uniform crossover operator as the best crossover operator.

Acknowledgments

The authors wish to acknowledge the approval and the support of this research study by the grant No. 7-14-1436-5 from the Deanship of Scientific Research in Northern Border University, Arar, KSA.

References

[1] Silberschatz, Abraham, *"Operating System Principles",* (7 ed.). Wiley-India. p. 237, 2006.
[2] Padua, David, *"Encyclopedia of Parallel Computing",* Springer. p. 524, 2011.
[3] Shibu, *"Intro To Embedded System",* (1st ed.). McGraw Hill Education. p. 446, 2009.
[4] Stuart, Brian L, *"Principles of operating systems",* (1st ed.). Cengage Learning. p. 446, 2008.
[5] E. W. Dijkstra, *"Cooperating sequential processes",* Technical Report EWD-123, Technological University, Eindhoven, The Netherlands, 1965.
[6] A. N. Haberman, "*Prevention of system deadlocks*", Communication of the ACM, Vol. 12, No. 7, July 1969, pp. 373-385.
[7] Shoshani and E. Coffman, *"Prevention, detection and recover from deadlock in multiprocess, multiple resource systems",* Technical Report 80, Princeton University, 1969.
[8] Belik F. *"An efficient deadlock avoidance technique",* IEEE Transactions on Computers. 1990 Jul;39(7):882-8.
[9] Yao, B., Yin, J., & Wu, W. , *"Deadlock Avoidance Based on Graph Theory",* International Journal of u-and e-Service, Science and Technology, 9(2), 353-362, 2016.
[10] Chen, M.; *"Policy based reinforcement learning approach Of Job shop scheduling with high level deadlock detection*"; MSc. Thesis; Iowa State University; Ames, Iowa; 2013.
[11] Choi, J. Y.; *"Design and comparative performance analysis of a heuristic-based parameterised Banker's algorithm using the CRL scheduling problems";* International Journal of Production Research; Vol. 53, No. 9, 2605–2616, http://dx.doi.org/10.1080/00207543.2014.970710; 2015.
[12] Luo, J. C.; and Et al.; *"Scheduling of deadlock and failure-prone automated manufacturing systems via hybrid heuristic search"*; International Journal of Production Research; http://dx.doi.org/10.1080/00207543.2017.1306132; 2017.
[13] Chen, M.; and Rabelo, L.; *"Deadlock-Detection via Reinforcement Learning"*; Ind Eng Manage; Vol.6: 215. doi:10.4172/2169-0316.1000215; 2017.
[14] Wu, Y.; and Et al.; *"Robust deadlock control for automated manufacturing systems with a single type of unreliable resources"*; Advances in Mechanical Engineering; Vol. 10(5) 1–14; DOI: 10.1177/1687814018772411; 2018.
[15] Ahmed, Nada Thanoon, and Narjis Mezaal Shati. *"A New Method for Solving Deadlock Using Genetic Algorithms."* International Journal of Advanced Research in Engineering and Technology 10.1, 2019.

[16] Smith, J., & Vavak, F. , *" Replacement strategies in steady state genetic algorithms: Static environments. Foundations of genetic algorithms"*, 5, 219-233, 1999.

[17] Poli, R., & Langdon, W. B., *"On the search properties of different crossover operators in genetic programming. Genetic Programming"*, 293-301, 1998.

[18] De Falco, I., Della Cioppa, A., & Tarantino, E., *"Mutation-based genetic algorithm: performance evaluation",* Applied Soft Computing, 1(4), 285-299, 2002.

Author Biography

Rabie Ahmed, Department of Computer Science, Faculty of computing & IT, Northern Border University, Rafha, Saudi Arabia. He received his PhD degree in Computer Science from Faculty of Science, Beni-Suef University, Egypt in 2012 after joint scholarship between Egypt and USA. His major research interests include Parallel and Distributed Computing, Artificial Intelligence and Machine Learning Techniques.

Taoufik Saidani, Department of Computer Science, Faculty of computing & IT, Northern Border University, Rafha, Saudi Arabia. He received his PhD degree in Computer Science and Engineering from Faculty of Science, of Monasstir, Tunisia in 2014. His major research interests include VLSI and embedded System in video and image compression, Digital image processing, Artificial Intelligence and Deep Learning.

Malek Rababa, Department of Computer Science, Faculty of computing & IT, Northern Border University, Rafha, Saudi Arabia. He received his Bachelor degree from Jerash Private University in 2004 and his master's degree from Al-Balqa Applied University in 2009. His major research interests include Artificial Intelligence.