A Heterogeneous Framework for the Global Parallelisation of Genetic Algorithms

Mohammad Hamdan
Department of Computer Science, Yarmouk University, Jordan

Abstract: *There is a big need for the parallelisation of genetic algorithms. In this paper, a heterogeneous framework for the global parallelisation of genetic algorithms is presented. The framework uses a static all-worker parallel programming paradigm based on collective communication. It follows the single program multiple data parallel programming model. It utilises the power of parallel machines by allowing multiple crossover and mutation operators being used within a single genetic algorithm. This mixture of operators can be applied to the strings of a population in parallel without changes to the canonical sequential genetic algorithm. These features help the parallel genetic algorithm in exploiting the search space efficiently and thoroughly when compared to the sequential genetic algorithm. The framework is instantiated with specific parameters to solve an NP-hard problem, the asymmetric travelling salesman problem. The results for the parallel genetic algorithm are very good in terms of solution quality. Also very good speedup and scalability results were achieved on the parallel machine.*

Keywords: *Genetic algorithms, parallel processing, parallel genetic algorithms, crossover, mutation, TSP.*

Received November 14, 2006; accepted February 28, 2007

1. Introduction

Genetic algorithm (GA) is part of what is called evolutionary computing that have been used successfully in solving search and optimization problems. They were invented by John Holland and his students in 1975 [13]. They have studied Darwin's theory about evolution and developed an algorithm that mimics selection in biological systems. However, there are few problems in GAs such as computational time and failure to achieve optimal or near optimal solutions.

In order to overcome these problems and improve GAs, the concept of parallel processing has been introduced in designing and implementing GAs. This is due to many reasons such as: GAs are inherently parallel, long execution time of GAs and the use of parallel machines for exploiting the search space thoroughly. There are different parallel approaches to GAs such as global parallelisation where the fitness function of the GA is evaluated in parallel using a dynamic master slave paradigm, island models where subpopulations reside on different processing nodes, fine grain where each string resides on a single node and in few cases hybrid of these approaches. This is clearly noted in the uses of parallel genetic algorithms for different application domains such as optimization [21], scheduling [17], image reconstruction [15], design problems [2] and analysis problems [14].

However, approaches in parallelizing GAs such as cellular and Island models [9] did alter the canonical structure of GA [19] and introduced new parameters in the parallel GA such as number of subpopulations, migration size and frequency, replacement strategy and islands topology.

Also these approaches had super linear results due to very strong selection pressure caused by excessive migration [24, 1]. Nonetheless, global parallelisation was useful only when evaluating the fitness function was computationally expensive. However, this approach relied on a dynamic master slave paradigm for data parallelism which suffers from extra communication overhead at the master node.

The main motivation of this work is to parallelize GAs without altering the canonical structure of sequential GA. Therefore an improved heterogeneous framework is presented for the global parallelisation of GAs. It is called heterogeneous since different crossover and mutation operators are applied to different subpopulation strings simultaneously. It was engineered by integrating the following techniques and operators into a truly novel heterogeneous framework: global single population, collective communication, static all-worker parallel programming model, multiple crossover operators, multiple mutation operators and application specific heuristics.

The rest of the paper is organized as follows. Section 2 presents the novel framework for parallelizing GAs. The model is customized in Section 3. In Section 4, the results are presented and discussed. Comparison with related work is outlined

in Section 5 and finally the work is summarized and future work is outlined in Section 6.

2. The Framework

The heterogeneous framework presented in this paper builds on previous work by the author. Initial study looked at the role of collective communication in a homogeneous framework for the global parallelisation of Gas [12].

In this paper, we are proposing a heterogeneous framework for the parallelization of the classical GA without altering its structure that uses collective communication in order to reduce communication overhead. Nonetheless, we are utilizing the power of Multiple Programs Multiple Data (MPMD) [20] approach by using different crossover and mutation operators in order to exploit the search space efficiently. However, the framework follows Single Program Multiple Data (SPMD) model of parallelism since all processing nodes have the same copy of code. Once the parallel program is executed each processing element will enter different parts of the code in order to execute different crossover and mutation operators.

The main features of the framework are as follows:

- Global parallelisation of GAs by using a static all-worker paradigm based on collective communication for parallel processing. In this model, all processing nodes apply the genetic operators to its local subpopulation. By static we mean divide the population into equally-sized number of subpopulations, send all strings for a given subpopulation using a big send message to the corresponding worker. The all worker paradigm differs from a standard dynamic master-slave paradigm in the utilization of the master node as it computes tasks in addition to communication.
- Collective communication rather than point to point communication due to the advantages of the former over the latter [8]. This approach fits very well with the static all-worker design for performing the computation due to the regular computation.
- Behaviour found in crossover and mutation operators. Also, this feature had good results on the scalability of the PGA as discussed in Section 4.
- Multiple crossover operators where it is possible to use a different crossover operator on different processing nodes.
- Multiple mutation operators where it is possible to use a different mutation operator on different processing nodes.
- Application-specific heuristics where it is possible to decode population strings then apply few heuristics to them then encode them back into GA strings.

The selection of either crossover or mutation operators is performed in the following manner. Assume there is C different crossover operators numbered from 1 to C and M different mutation operators numbered from 1 to M. Then worker i will use the j^{th} crossover operator and the k^{th} mutation operator where $j = i \% C$ and $k = i \% M$. Figure 1 illustrates the process.

The new framework utilises the power of parallel computing as it uses different crossover operators on different parts of the population by using different processors in the parallel environment. This has the advantage of the possibility of introducing a bigger number of new strings every generation when compared with using only one crossover operator especially when the different crossover operators are applied to the same parents. However, it is not necessarily that the same parents will be copied across different processors as it depends on the similarity of strings in the population.

The framework consists of two parts; the root worker and the secondary workers. Assume there are N nodes in the cluster numbered from 1 to N then node number 1 will be root worker and remaining nodes are secondary workers. The root and secondary worker are described in the following subsections.

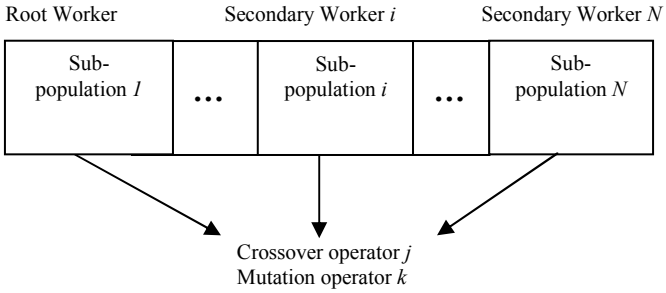

Figure 1. Applying different crossover and mutation operators to the subpopulations.

2.1. The Root Worker

Figure 2 illustrates the pseudo-code for the root worker that is going to be executed only on the root node. It is called *root worker* rather than master node as it not only controls the behaviour of the processing nodes but also performs work similar to the secondary workers. It is possible for the root worker to process tasks since the common communication bottleneck in standard master slave paradigm has been reduced by the use of collective communication rather than point to point sends and receives.

The role of the root worker is to generate the initial population where all strings in the population are generated randomly according to the representation system used. Global parent selection is then carried out where parents are selected from the current population. The parents are divided across all processors evenly. Each processor (secondary worker) will get its corresponding piece of the population. The

root worker will then select and a crossover operator and a mutation and apply it to its local subpopulation.

After collecting the new subpopulations it will repeat the process until the PGA converges. It is important to note that this framework preserves the behaviour of the canonical GA as it performs global selection only on the root worker.

1. *Generate initial population randomly.*
2. *Evaluate initial population.*
3. *Apply parent selection operator and generate candidate parents.*
4. *Divide the single population into equally-sized number of subpopulations.*
5. *Use a scatter operation in order to send to every worker its corresponding subpopulation. Keep one subpopulation for local processing.*
6. *Select and apply crossover operator to the local subpopulation.*
7. *Selectand apply mutation operator to the local subpopulation.*
8. *Decode strings apply problem-specific heuristics to the local subpopulation and encode strings.*
9. *Evaluate local subpopulation.*
10. *Use a gather operation in order to collect the new subpopulations from the slave workers.*
11. *Apply an elitism operation on the gathered single population.*
12. *Repeat steps 3 through 12 until convergence criteria are met.*
13. *Broadcast Termination message to secondary Workers.*

Figure 2. Pseudo-code for the root worker.

2.2. Secondary Workers

Figure 3 illustrates the pseudo-code for the secondary workers. This code will be executed on all processing nodes except the root node. The role of the secondary workers is to receive its corresponding piece of the big population then perform the required operators on the population. The selected crossover and mutation operators will be applied to all strings of the worker's subpopulation. Different workers will use different crossover and mutation operators. Once the termination message is received from root worker all secondary workers will stop.

3. Using and Applying the Framework

The heterogeneous framework can be simply used by specifying the following:

- Number of strings, probabilities for crossover and mutation.
- Representation technique that best suits the application which could be binary, arithmetic, float or permutation.
- Encoding and decoding which simply explains how to encode a possible solution of the problem using the representation scheme and how to decode a population string into a possible problem solution.
- Evaluation of strings by providing the fitness function that can be used to evaluate the strings of the population.
- Crossover operators: the user needs to provide the framework with as many as possible different crossover operators that work on the given representation scheme in order to exploit the search space thoroughly.

1. *Receive subpopulation from root worker using a global scatter operation.*
2. *Select and apply crossover operator to local subpopulation.*
3. *Select and apply mutation operator to local subpopulation.*
4. *Decode Strings, apply problem-specific heuristic to local subpopulation and encode strings.*
5. *Evaluate subpopulation.*
6. *Send subpopulation to root worker using a global gather operation.*
7. *Repeat steps 2 through 9 until a termination message is received from root worker.*

Figure 3. Pseudo-code for the secondary workers.

- Mutation operators: the user needs to provide the framework with as many as possible different mutation operators that work on the given representation scheme in order to prevent premature convergence.
- Heuristics: these are problem-specific heuristics that can be used in order to help the GA. They may work on the coded or decoded strings. It is up to the user to use this feature or not.
- Number of workers: this includes root and secondary workers.
- In order to test the framework, we have instantiated the framework with the needed operators in order to solve a combinatorial problem: the asymmetric travelling salesman problem. Path encoding was used and the fitness function is simply 1/(tour length).

The rest of the operators are described below:

- The following crossover operators were placed on different nodes [16]: order, modified order, cycle, heuristic, alternating position, edge recombination and partial matched.
- The following mutation operators were placed on different nodes [16]: reciprocal exchange, inversion, insertion, displacement, boundary and uniform mutation.
- TSP-heuristics: modified 2-opt step where N attempts are performed to shorten the tour and modified or-opt step where all possible subtours of length 3, 2 or 1 city are relocated into all possible pairs of cities. The new tour is taken if it results in a shorter tour.

4. Results

To verify the instantiated framework, the following five anti-symmetric TSP instances: *p43, ftv44, ry48p, ft53* and *kro124* were taken from TSPLIB[22] and used as input data to the PGA. Two types of experiments were conducted where each used a different type of GA termination: 1) run the PGA for a fixed number of generations i.e., 5000 iterations and 2) stop the GA when the best solution found so far has not improved

for a given number of generations. The machine used for the experiments had the following technical details: IBM xSeries 335 with Intel Xeon 2.4 GHz processors, 1 GB RAM 512 KB L2 cache and a 1 GB Myrinet for communication. Myrinet is known for its low latency communication.

The first set of experiments was needed in order to study the scalability behaviour of the framework. Two major points were addressed in the experiments: increasing the number of processors and the use of TSP instances that have bigger number of cities. The second set of experiments were needed in order to find out if the parallel GA will converge after evolving less or more populations when compared to the sequential GA. This is needed in order to prove that the speedup reported is not due to generating less number of generation caused by excessive migration or other issues as found in [1].

For both cases, the global population size is 384 individuals, used fitness proportionate selection over the global population, applied the selected crossover operator to 80% of the population, applied the selected mutation operator to 0.05% of the population and applied the modified 2-opt and or-opt to 25% of the population.

The following crossover operators were used on different processors: order, modified order, cycle, heuristic, alternating position, edge recombination and partial matched. Also the following different mutation operators were used on different processors: reciprocal exchange, inversion, insertion, displacement, boundary and uniform mutation. We report results for average number of generations needed for the GA to converge and percentage quality difference for the best, average and the worst solutions found from the optimal known solution to each TSP instance. Also speedup of the parallel GA over the sequential GA is presented. The heuristic crossover operator and insertion mutation operator were used for the SGA. All results were averaged over 50 runs.

Table 1 shows the results for the first case. In the Table, *#P* stands for number of processors. The sequential GA is when *#P = 1*. It is clear from the results for all TSP instances that the average quality solution of the parallel GA is better than the sequential GA. This shows that the PGA has explored the search space in a better way and managed to find better solutions to the problem than the sequential GA. This clearly noted for the last two TSP instances *ft53* and *kro124* as the complexity of the problem increases due to the bigger number of cities. For *ft53*, the average quality difference solution is 3.39 for the SGA and improves to 2.35 using 32 nodes for the PGA. Also for *kro124* the SGA failed to find the optimal solution. The best quality difference solution was 2.06 while few instances of the PGA manage to find the optimal solution.

The possible reason for the PGA outperforming the SGA regarding quality of solution can be due to the following: by having multiple processors, each processor will be responsible for generating the initial seed for its random number generator. It is expected that each processor will have different sequences of random numbers and as we know GA operators depend heavily on a good random number generator. By having different processors and thus different random numbers the possibility of exploring the search should be better in PGA.

In Figure 4, we show the speedup results for various TSP instances and the scalability of the framework. It is clear from the results that speedup improves as the PGA uses more processors. This supports the claim that the framework is scalable and it is interesting to note that speedup also improves as the TSP instance has a bigger number of cities. This is clearly noted as we got a speedup of 12. 13 using 32 processors for the *p43* instance that has only 43 cities while we had a speedup of 26.48 using 32 processors for the *kro124* instance that has about 100 cities to be visited. This is due to the increase in computation over communication.

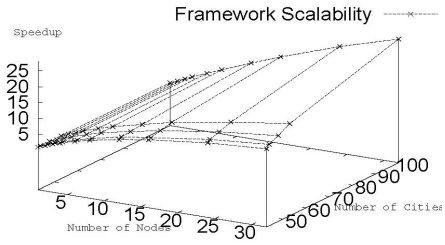

Figure 4. Framework scalability.

The results for the second case are shown in Table 2. As mentioned earlier the aim of this experiment to show that the PGA does not necessarily need less number of iterations to converge when compared with the SGA. For the first two TSP instances the PGA needed nearly the same or slightly less number of iterations to converge.

For the remaining TSP instances the PGA needed more iteration to converge. Therefore, the PGA generated more populations and this requires more computation than the SGA. Due to the efficiency of the framework it took less time to compute even though there were more generations to be evolved. This insures that premature convergence is avoided and the search space is exploited properly by evolving as many generations as possible.

5. Comparison with Related Work

The reader may refer to [5, 18, 23, 3] for detailed surveys about the parallelization of GAs. In the following paragraphs, more recent and related techniques for parallel GAs are compared to the framework presented in this paper.

Table 1. Results for case 1.

p43										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	655.26	570.58	726.8	518.88	527.56	480.8	626.64	731.7	553.9	497.46
Speedup	1	1.87	2.67	3.41	4.73	5.75	7.51	8.95	10.99	12.13
Best	0	0	0	0	0	0	0	0	0	0
Avg	0.017	0.019	0.019	0.018	0.019	0.017	0.017	0.014	0.014	0.02
Worst	0.053	0.053	0.053	0.053	0.053	0.034	0.017	0.036	0.036	0.053
ftv44										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	814.26	734.92	848.4	908.22	770.4	893.82	837.88	818.02	682.08	951.16
Speedup	1	1.84	2.71	3.38	4.77	5.97	8.04	9.25	11.72	13.17
Best	0	0	0	0	0	0	0	0	0	0
Avg	2.23	2.18	2.17	١,٩٤	١,٩٩	١,٩٤	٢,٢٢	٢,٠٦	٢,٢٧	١,٨
Worst	7.01	٥,٨٩	٦,٧٦	٦,٧٦	٦,٨٨	٦,٨٨	٤,٩	٦,٨٨	٦,٨٨	٤,٩٠
RY48P										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	897.7	٧٧٥,٧٨	٧٧٤,٦	٧١١,٠٤	٨٧٨,٨	٧٦٩,٩٤	٨٦٥,٦٤	٧٨١,٨٢	٦٠٠,٩٦	٧٤٣,٠٤
Speedup	1	1.92	2.79	3.57	5.06	6.39	8.28	10.04	12.72	14.52
Best	0	0	0	0	0	0	0	0	0	0
Avg	0.79	0.65	0.78	0.62	0.64	0.5	0.66	0.69	0.54	0.66
Worst	2.69	2.67	2.69	1.84	2.63	1.91	2.63	2.22	1.84	2.69
ft53										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	1065.16	985.74	1130.66	1169.7	1258.2	1155.24	1089.34	972.78	1232.92	1083.68
Speedup	1	1.94	2.71	3.66	5.2	6.54	8.73	10.88	13.97	16.54
Best	0	0	0	0	0	0	0	0	0	0
Avg	3.39	2.4	2.76	2.33	2.45	2.28	2.42	2.09	2.40	2.35
Worst	10.4	7.14	8.99	8.05	10.38	8.86	10.33	7.6	9.85	9.14
kro124										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	1776.73	1710.54	1849.66	1786.82	1884.88	1746.6	1806.22	1773.68	1842.91	1920.37
Speedup	1	1.99	2.99	3.99	5.86	7.76	11.36	14.82	21.08	26.48
Best	2.06	0.03	1.92	0	0	0	0	0.03	0	0.37
Avg	5.84	3.81	4.62	3.64	4.07	3.73	4.027	4.26	4.09	3.84
Worst	13.22	7.34	9.31	7.7	8.15	7.79	9.63	8.74	7.83	9.42

Table 2. Results for case 2.

p43										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	280.96	241.66	250.72	274.8	258.5	224	255.82	249.44	247.56	266.78
Speedup	1	1.96	2.84	3.44	4.89	6.49	7.84	9.5	١٢,٤٣	١١,٥٦
Best	0	0	0	0	0	0	0	0	0	0
Avg	0.024	0.026	0.023	0.02	٠,٠٢٤	٠,٠٢٦	٠,٠٢٧	٠,٠٢٦	٠,٠٣٣	٠,٠٣١
Worst	0.053	0.053	٠,٠٥٣	٠,٠٥٣	٠,٠٥٣	٠,١٠٧	٠,٠٨٩	٠,٠٧١	٠,٢٣	٠,٠٨٩
ftv44										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	252.04	229.52	242.74	223.12	240.28	227.2	285.42	241.14	241.84	265.12
Speedup	1	1.96	2.81	3.71	4.98	6.34	7.39	9.9	11.99	13.39
Best	0	0	0	0	0	1.3	0	0	0	0
Avg	3.07	3.28	2.83	٣,٣٢	٢,٨٧	٢,٥٩	٣,١٢	٢,٨٤	٢,٦١	٢,٦١
Worst	8.49	٦,٧٦	٧,٣٢	٩,١١	٦,٧٦	٦,٨٨	٧,٩٤	٧,٨١	٦,٨٨	٧,٣٢
RY48P										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	324.96	364.48	٣٩٣,٢٦	٣٢٣,٣٦	٣٦٦,١٢	٣٤٣,٥٢	٣٨٦,٨٨	٣٩٠,٩٢	٣٧١,٥٢	٣٧٩,٨٨
Speedup	1	1.79	2.52	3.31	٤,٥٧	٥,٧٤	٢,٢٧	٩,٤٧	١٢,٧	١٣,١٧
Best	0	0	0	0	0	0	0	0	0	0
Avg	1.45	٠,٩٧	١,٠٧	١,٠٦	١,٠٩	٠,٨٤	٠,٩٠	٠,٧٧	١,١٣	٠,٩١
Worst	4.17	٢,٢٩	٣,٢٥	٣,٧٤	٢,٨٢	٢,٤٦	٣,٦١	٢,٣٣	٣,١٣	٣,٨٢
ft53										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	341.26	364.12	370.54	369.96	398.28	394.34	397	352.74	406.44	383.74
Speedup	1	1.85	2.65	3.44	4.7	6.05	8.08	10.4	12.6	15.32
Best	0	0.06	0.15	0	0	0	0	0	0	0.06
Avg	5.75	4.71	4.12	3.99	3.8	4.33	4.3	3.78	3.71	3.75
Worst	14.35	13.4	10.72	11.57	13.51	11.34	12.14	11.39	10.38	10.82
kro124										
#P	1	2	3	4	6	8	12	16	24	32
Avg Gen	785.52	٩٨٧,٤٦	٩٢٤,٣	٩٥١,٧٨	٨٨١,٥٧	٩٥١,١	٩٤٢,٣	٩٢٥,٥٤	١٠٠٨,٠٥	١٠٠٦,٠٦
Speedup	1	١,٦٦	٢,٦٤	٣,٤١	٥,١٩	٦,٦٤	٩,٨٠	١٣,١	١٧,٤٣	٢١,٧
Best	1.32	٠,٠٣	١,٥٥	٠,٥١	٠	١,١٩	١,٦٤	١,٥٦	٠,١٨	٠,٠٣
Avg	7.05	٥,٠٦	٦,٢١	٤,٨٥	٥,٤٣	٥,٥٣	٥,١٠	٤,٧٤	٤,٩٢	٥,٠٥
Worst	11.62	١٠,٩	١٢,٢١	٨,٠٢	١٠,٠٣٩	١١,٨٥	٩,٦٥	٩,٣٤	٨,٣٣	١٠,٤٧

The Hy3 [6] is a new model for optimization in continuous domains. The model is based on a fixed number of eight subpopulations residing in a cube topology. Therefore, the model follows a migration based island model of GAs. The authors claim that it is possible to use different crossover operators on different island but it is not clear from the paper what crossover operators were used and their role in exploiting the search space. Results are discussed for optimizing functions in continuous domains and no examples for combinatorial problems such as TSP.

Nonetheless, in the framework for studying PGAs [4], it was pointed out that PGAs almost always outperform SGAs. Also in the future work of the authors they mentioned that it would be interesting to study the importance of using different operators in every island which we have managed to implement in this paper. In [26], a description of a parallel distributed implementation of genetic programming that can exploit the inherent parallelism in semi-isolated subpopulations is given. The work differs from our framework as it looks at migration-based systems that rely on genetic programming techniques and not GAs.

Other interesting studies looked at PGAs for the scheduling problem [17] where the GA was divided into multiple "demes". Also the PGA for TSP discussed in [25] was based on an island model that uses distinct sub-population. The parallelisation was done through a standard master-slave paradigm. Results are shown for symmetric TSPs and not Anti-symmetric TSPs as shown in this paper. Also no major speedup results were reported due to the communication overhead caused by the dynamic master-slave paradigm. However, our framework did not suffer from this communication overhead as it uses collective communication. Nonetheless, the authors pointed out that it would be interesting to implement the PGA using an MPMD paradigm which was achieved in the framework presented in this paper.

There are parallel implementations of genetic programming based on the cellular model as presented in [11]. This approach completely differs from our framework as it makes major changes to the canonical genetic algorithm. Also there are hardware implementations for GAs such as the diffusion model presented in [10]. It is architecture for distributed GAs based on a massively parallel GA (cellular GA) and implemented in hardware as an alternative to island-based GAs.

Regarding sequential GAs, multiple mutation operators are selected and applied in an adaptive approach to the strings of the population [27]. The work differs from our framework as it cannot apply different mutation operators to the population simultaneously but uses one mutation operator then changes to another one according to the average fitness of the strings.

We think that the framework presented in this paper differs from previous ones as it is novel in the integration of collective communication, an all-slave paradigm for parallel programming, selection over global population, multiple crossover operators and multiple mutation operators in a heterogeneous framework for the parallelization of GAs without changes to its canonical design. No new parameters are needed to control the parallelization such as migration rate, size, and frequency.

6. Conclusions and Future Work

The presented heterogeneous framework truly utilized the power of parallel machines. It was engineered using multiple crossover and mutation operators and an all-slave paradigm based on collective communication. It managed in reducing the execution time of sequential GAs without changes to its basic structure. Also, it managed in exploring the search space efficiently and thoroughly. The framework was instantiated to solve a non trivial combinatorial problem: ATSP. The results obtained showed that the framework is capable of reducing the execution time and managed in finding optimal and near optimal solutions to ATSP.

In the future work, the framework will be instantiated with problem parameters in order to solve other applications such as scheduling and optimization. Nonetheless, it is interesting to generate a performance model that can predict the behaviour of the framework in a similar way to the cost model in [7]. Also, it might be interesting to modify the SGA where multiple crossover and mutation operators can be applied to different strings of the subpopulation for further comparison.

7. Acknowledgments

The author would like to thank the DAAD organization for the research visit grant to Munster University in Germany. Also, thanks to professor Sergie Gorlatch and his group for hosting the visit and giving access to their parallel computing cluster during the visit and after return to home country.

References

[1] Affenzeller M. and Wagner S., "Sasegasa: A New Generic Parallel Evolutionary Algorithm for Achieving Highest Quality Results," *Journal of Heuristics*, vol. 10, no. 3, pp. 243-267, 2004.

[2] Alba E. and Chicano F., "On the Behaviour of Parallel Genetic Algorithms for Optimal Placement of Antennae in Telecommunications," *International Journal of Foundations of Computer Science*, vol. 16, no. 2, pp. 343-359, 2005.

[3] Alba E. and Tomassini M., "Parallelism and Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 443-462, 2002.

[4] Alba E. and Troya J., "Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms," *Future Generation Computer Systems*, vol. 17, pp. 451-465, 2001.

[5] Alba E. and Troya J., "A Survey of Parallel Distributed Genetic Algorithms," *Complexity*, vol. 4, no. 4, pp. 31-52, 1999.

[6] Alba A., Luna F., Nebro A., and Troya J., "Parallel Heterogeneous Genetic Algorithms for Continuous Optimization," *Parallel Computing*, vol. 30, pp. 699-719, 2004.

[7] Bazterra V., Cuma M., Ferraro M., and Facelli J., "A General Framework to Understand Parallel Performance in Heterogeneous Cluster: Analysis of a New Adaptive Parallel Genetic Algorithm," *Journal of Parallel and Distributed Computing*, vol. 65, no. 1, pp. 48-57, 2005.

[8] Calvin C. and Colombet L., "Performance Evaluation and Modelling of Collective Communications on Cray t3d," *Parallel Computing*, vol. 22, pp. 1413-1427, 1996

[9] Cantu-Paz E., *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic, 2000.

[10] Eklund S., "A Massively Parallel Architecture for Distributed Genetic Algorithms," *Parallel Computing*, vol. 30, pp. 647-676, 2004.

[11] Folino C. and Spezzano G., "A Scalable Cellular Implementation of Parallel Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 37-53, 2003.

[12] Hamdan M., "Collective Communication, Multiple Mutation Operators and the Global Parallelisation of Genetic Algorithms," *in Proceedings of the 5th International Conference of Recent Advances in Soft Computing (RASC2004)*, Nottingham, UK, pp. 300-305, 2004.

[13] Holland J., *Adaptation in Natural and Artificial Systems*, the University of Michigan Press, 1975.

[14] Kai Xu S. and Mancini R., "A Scalable Parallel Genetic Algorithm for X-Ray Spectroscopic Analysis," *in Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 811-816, 2005.

[15] Knoll P. and Mirazaei S., "Validation of a Parallel Genetic Algorithm for Image Reconstruction from Projections," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 356-359, 2003.

[16] Michalewicz Z., *Genetic Algorithms+Data Structures= Evolution Programs*, Springer-Verlag, Berlin, 1996.

[17] Moore M., "An Accurate Parallel Genetic Algorithm to Schedule Tasks on a Cluster," *Parallel Computing*, vol. 30, no. 5-6, 2004.

[18] Nowostawski M. and Poli R., "Review and Taxonomy of Parallel Genetic Algorithms," *Technical Report CSRP-99-11*, the University of Birmingham, UK, 1999.

[19] Pereira C. and Lapa C., "Parallel Island Genetic Algorithm Applied to a Nuclear Power Plant Auxiliary Feed Water System Surveillance Tests Policy Optimization," *Annals of Nuclear Energy*, vol. 30, pp. 1665-1675, 2003.

[20] Quinn M., *Parallel Programming in C with MPI and OpenMP*, McGraw Hill, New York, USA, 2003.

[21] Rahul D. and Dutta A., "Optimization of frp Composites Against Impact Induced Failure Using Island Parallel Genetic Algorithm," *Composites Science and Technology*, vol. 65, no. 13, pp. 2003-2013, 2005.

[22] Reinelt G., "Tsplib-a Travelling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376-384, 1991.

[23] Rivera W., "Scalable Parallel Genetic Algorithms," Artificial Intelligence Review, vol. 16, pp. 153-168, 2001.

[24] Sang-Keon Oh C. and Lee J., "Balancing the Selection Pressures and Migration Schemes in Parallel Genetic Algorithms for Planning Multiple Paths," *in IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3314-3319, 2001.

[25] Sena G., Megherbi D., and Isern G., "Implementation of a Parallel Genetic Algorithm on A Cluster of Workstations: Travelling Salesman Problem, A Case Study," *Future Generation Computer Systems*, vol. 17, no. 4, pp. 477-488, 2001.

[26] Tanev T. and Ono K., "Scalable Architecture for Parallel Distributed Implementation of Genetic Programming on Network of Workstation," *Journal of Systems Architecture*, vol. 47, no. 7, pp. 557-572, 2001.

[27] Tzung-Pei H. and Chen W., "Simultaneously Applying Multiple Mutation Operators in Genetic Algorithms," *Journal of Heuristics*, vol. 6, no. 4, pp. 439-455, 2000.

Mohammad Hamdan received his PhD from Heriot-Watt University in 2000. His PhD thesis title was "A Framework for Parallel Programming Using Algorithmic Skeletons". Since February 2000, he has been working as an assistant professor in the Department of Computer Science. In September 2002 he became the assistant dean in the Faculty of Information Technology. In September 2005, he became the chairman of Computer Science Department. He is a senior member in IEEE. Since January 2002, he became the activity secretary for Jordan IEEE executive committee and in May 2006 he became the chair for IEEE Computer Chapter in Jordan. His research interests are parallel processing, genetic algorithms, and the parallelisation of genetic algorithms.