



YAŞAR UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MASTER THESIS

PARALLEL EVOLUTIONARY ALGORITHMS
FOR
QUADRATIC ASSIGNMENT PROBLEM

ALPER KIZIL

THESIS ADVISOR: ASST. PROF. DR. KORHAN KARABULUT

COMPUTER ENGINEERING MASTERS PROGRAM

PRESENTATION DATE: 10.08.2017

BORNOVA / İZMİR
AUGUST 2017

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Jury Members:

Asst. Prof Dr. Korhan KARABULUT
Yaşar University

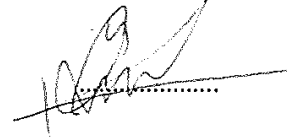
Signature:



Prof. Dr. Aybars UĞUR
Ege University



Asst. Prof Dr. Dindar ÖZ
Yaşar University



Prof. Dr. Cüneyt GÜZELİŞ
Director of the Graduate School

ABSTRACT

PARALLEL EVOLUTIONARY ALGORITHMS FOR QUADRATIC ASSIGNMENT PROBLEM

KIZIL, Alper

MSc. in Computer Engineering

Advisor: Asst. Prof. Dr. Korhan KARABULUT

August 2017

Quadratic Assignment Problem (QAP) is one of the most difficult combinatorial problems. There are many approaches proposed in literature to solve QAP. Genetic algorithms are nature inspired metaheuristics which can create good enough solutions in reasonable time. But for large size problems, they may be insufficient. This is due to search space they operate becomes too large and algorithm starts to miss out some parts. In this thesis, island model genetic algorithms are used to enhance a standard sequential genetic algorithm in terms of solution quality. Results show that, even with the most basic 2 island model, the proposed algorithm is able to obtain 3 times better results when solving QAP instances. The proposed algorithm is tested and fine-tuned for some of the parameters to enhance the algorithm even further. It is also observed that, different parameters effect solution quality. Ultimately, the proposed algorithm is able to come up with good enough configurations that can solve QAP instances up to 3% gap compared to the best-known solutions in the literature.

Key Words: Quadratic Assignment Problem, Metaheuristics, Genetic Algorithms, Island Model, Parallelism, Combinatorial Optimization.

ÖZ

İKİNCİ DERECE ATAMA PROBLEMİ İÇİN PARALEL EVRİMSEL ALGORİTMALAR

KIZIL, Alper

Yüksek Lisans Tezi, Bilgisayar Mühendisliği

Danışman: Yrd. Doç. Dr. Korhan KARABULUT

Ağustos 2017

Karesel Atama Problemi (KAP) en zor birleşimsel problemlerden birisidir. Literatürde KAP'ni çözmek için birçok yaklaşım önerilmiştir. Genetik algoritmalar doğadan ilham alan, makul bir zaman aralığında iyi sayılabilecek çözümler üreten metasezgisellerdir. Ancak geniş boyutlu problemler için yetersiz kalabilirler. Bunun sebebi bu algoritmaların üzerinde çalıştığı arama uzayının çok genişlemesi ve algoritmanın bu arama uzayının belirli bölümlerini gözden kaçırabilmesidir. Bu tez çalışmasında, ada modeli olarak tanımlanan bir modeli standart sıralı genetik algoritmayı çözüm kalitesi yönünden geliştirmek için kullanılmıştır. Sonuçlar, önerilen algoritmanın, en temel 2-adalı model ile dahi, KAP örneklerinde 3 kat daha iyi çözüm bulabildiğini göstermiştir. Önerilen algoritma test edilmiş ve bazı parametrelerine ince ayar yapılarak çözüm kalitesi daha da artırılmıştır. Ayrıca değişik parametrelerin sonuç kalitesine etkileri gözlemlenmiştir. Sonuçta, önerilen algoritma yeterince iyi konfigürasyonlarla KAP örneklerini literatürdeki en iyi çözümlere %3 yakınlıkta çözebilme düzeyine çıkabilmiştir.

Anahtar Kelimeler: Karesel Atama Problemi, Metasezgiseller, Genetik Algoritmalar, Ada Modeli, Paralleleştirme, Kombinatoryal Optimizasyon.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Asst. Prof. Dr KARABULUT for his guidance and patience during this study.

I would like to express my enduring love to my parents, who are always supportive, loving, motivative and caring to me in every possible way in my life.

I would like to thank all my co-workers and friends who always supported me and helped me with best of their abilities.

Alper KIZIL

Izmir, 2017



TEXT OF OATH

I declare and honestly confirm that my study, titled “PARALLEL EVOLUTIONARY ALGORITHMS FOR QUADRATIC ASSIGNMENT PROBLEM” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Alper KIZIL

Signature

.....

September 5, 2017

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vii
ACKNOWLEDGEMENTS	ix
TEXT OF OATH	xi
TABLE OF CONTENTS	xiii
LIST OF FIGURES	xv
LIST OF TABLES	xvi
SYMBOLS AND ABBREVIATIONS	xix
CHAPTER 1 INTRODUCTION	1
1.1. CLASSIFICATION OF PROBLEMS	2
1.2. QUADRATIC ASSIGNMENT PROBLEM AS AN NP- HARD PROBLEM	4
1.3. METAHEURISTICS FOR NP HARD PROBLEMS	6
1.4. NO FREE LUNCH THEOREM	7
1.5. GENETIC ALGORITHMS AS METAHEURISTICS	8
1.5.1. GA PROPERTIES	9
1.6. PARALLEL COMPUTING.....	20
1.6.1. FLYNN’S TAXONOMY	22
1.6.2. PARALLEL COMPUTING PERFORMANCE CRITERIA.....	22
CHAPTER 2 PARALLEL EVOLUTIONARY ALGORITHMS	24
2.1. PARALLEL GA MODELS	24
2.1.1. PARALLEL INDEPENDENT RUNS.....	24
2.1.2. MASTER-SLAVE MODEL.....	25
2.1.3. ISLAND MODEL.....	26
2.1.4. CELLULAR MODEL.....	28
2.1.5. HYBRID MODELS	29
2.2. MEASURING PERFORMANCE IN PARALLEL GAS	30
2.2.1. COMPUTATION TIME GAIN.....	30

2.2.2. SOLUTION QUALITY GAIN.....	31
CHAPTER 3 LITERATURE REVIEW	32
3.1. LITERATURE REVIEW ON QUADRATIC ASSIGNMENT PROBLEM.....	32
3.1.1. EXACT ALGORITHMS IN THE LITERATURE	35
3.1.2. METAHEURISTIC ALGORITHMS IN LITERATURE	37
3.1.3. CURRENT SITUATION IN QAP	53
3.2. LITERATURE REVIEW ON PARALLELIZATION OF GA	55
CHAPTER 4 AN ISLAND MODEL PARALLEL GA FOR QAP	63
4.1. FORMULATION AND SOLUTION REPRESENTATION	63
4.2. IMPLEMENTATION OF THE ALGORITHM	63
4.3. ALGORITHM TERMINATION.....	67
4.4. CODING AND BENCHMARKING THE ALGORITHM.....	68
CHAPTER 5 CONCLUSIONS AND FUTURE WORK.....	95
REFERENCES	97

LIST OF FIGURES

Figure 1.1. A Pseudo – code for GA (Eiben & Smith, 2003)	11
Figure 1.2. An Iteration Cycle of GA (Eiben & Smith, 2003).....	11
Figure 1.3. Sample swap mutation between two alleles	17
Figure 1.4. Insert mutation between two alleles	17
Figure 1.5. Scramble mutation.....	17
Figure 1.6. Inversion mutation.....	17
Figure 1.7. Single Point Crossover	18
Figure 1.8. Partially Mapped Crossover	19
Figure 1.9. Edge Crossover.....	20
Figure 1.10. List Order Crossover	20
Figure 1.11. Serial Processing (Barney, 2009)	21
Figure 1.12. Parallel Processing (Barney, 2009).....	21
Figure 2.13. Master-Slave Model	25
Figure 2.14. A unidirectional ring, a torus and a complete graph (Sudholt, 2015).....	27
Figure 2.15. Graph representation of the cellular model. (Sudholt, 2015)	29
Figure 4.1. Island GA Pseudocode	66
Figure 4.2. Number of islands versus mean RPC graph for Tai30a instance	73
Figure 4.3. Number of islands versus mean RPC graph for Tai40a instance	74
Figure 4.4. Number of islands versus mean RPC graph for Tai50a instance	76
Figure 4.5. Number of islands versus mean RPC graph for Tai60a instance	77
Figure 4.6. Number of islands versus mean RPC graph for Tai80a instance	78
Figure 4.7. Parallelization efficiency of Island Model over sequential Algorithm.....	80

LIST OF TABLES

Table 1.1. Tim Sort average number of comparisons depending on instance size	2
Table 1.2. Brute force TSP Solution average number of permutations to test depending on instance size	3
Table 1.3 Most efficient TSP Solution average number of permutations to test depending on instance size	3
Table 1.4 Simple GA Properties	9
Table 3.5. QAPLIB Instances with optimum values unknown & Best-known metaheuristics. (Burkard, Karisch, & Rendl, 1997).....	54
Table 4.1. Parallel Island GA Parameters Initial Experiment.....	66
Table 4.2. Sequential GA Parameters	67
Table 4.3. Parallel Island GA Preliminary Test – 1 Parameters	69
Table 4.4. Parallel Island GA Preliminary Test-1, Top 13 Most Successful Configurations out of 96, based on average RPC	69
Table 4.5. Parallel Island GA Preliminary Test -2 Parameters	71
Table 4.6 Parallel Island GA Preliminary Test -2 Results.....	71
Table 4.7 Parallel Island GA Optimizing Number of Islands.....	72
Table 4.8 Number of Islands Experiment Results with Tai30a data set	72
Table 4.9 Number of Islands Experiment Results with Tai40a data set	73
Table 4.10 Number of Islands Experiment Results with Tai50a data set	75
Table 4.11 Number of Islands Experiment Results with Tai60a data set	76
Table 4.12 Number of Islands Experiment Results with Tai80a data set	77
Table 4.13 Parallelization Efficiency (%) on Taillard samples.	79
Table 4.14 Unidirectional – Bidirectional Ring Migration Comparison on Tai30a	81
Table 4.15 Unidirectional – Bidirectional Ring Migration Comparison on Tai40a	81
Table 4.16 Unidirectional – Bidirectional Ring Migration Comparison on Tai50a	82
Table 4.17 Unidirectional – Bidirectional Ring Migration Comparison on Tai60a	82
Table 4.18 Unidirectional – Bidirectional Ring Migration Comparison on Tai80a	82

Table 4.19 : Optimizing Other Parameters on Taillard a Instances – Possible Configurations	83
Table 4.20 Taillard a Samples Optimizing Number of Immigrants.....	84
Table 4.21 : Taillard a Samples Optimizing Immigration Time (Number of Iterations)	85
Table 4.22 Taillard a Samples Optimizing Number of Elites per Island	86
Table 4.23 : Taillard a Samples Best Parameters found on Taillard a Data Set.....	87
Table 4.24 : Taillard a Instances Optimum Configuration Results	88
Table 4.25 : Burkard Instance Set Optimizing Number of Islands	89
Table 4.26 : Steinberg Instance Set Optimizing Number of Islands	90
Table 4.27 : Nugent Instance Set Optimizing Number of Islands.....	92

SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS:

QAP Quadratic Assignment Problem

EA Evolutionary Algorithm

GA Genetic Algorithm

SA Simulated Annealing

AC Ant Colony

TS Tabu Search



CHAPTER 1

INTRODUCTION

Optimization is the process of selecting the best element out of a set of elements. This is a concept widely used in real life. For instance, when we want to go from one place to another, we think of alternative routes and select the best possible in terms of money and time. Similarly, a company producing manufactured products wants to maximize their profit by minimizing raw material cost.

Another and very famous example can be found in business of shipping goods. Package delivery companies usually deliver many packages to their destinations daily. Suppose that a truck owned by the company delivers supplies to several locations. To maximize their profit, they must reduce their expenditures. The simplest way to accomplish this is to select a route in such a way that, truck will travel the least distance, therefore it will consume least fuel. Distance travelled in a possible route can be calculated by accumulatively adding distances in each step for one stop from another.

A shipping company can use a computer to calculate all possible route permutations and select the permutation with the least cost. Tricky part here is the number of stops along the way. As the number of stops increase, number of possible permutations will increase exponentially to the point where an efficient calculation becomes impossible, because truck will start from the depot with n possible next stops to select and any remaining $n - 1$ possible stops may be selected as the second stops and so on. This will make number of calculations $n!$ for n stops. For small numbers like 10, there will be 3.628.800 possible alternate routes, which can be handled by a computer. For larger number of stops like 20, we will have 2.432.902.008.176.640.000 alternate routes and calculation will be overwhelming.

The mentioned problem is known as the classical “Traveling Salesman Problem” in computer science and it has a wide area of research. There are many other problems like Traveling salesman which share common characteristics. That is, as the input size of problem grows, number of possible solutions grow exponentially. One other

intriguing fact is that, some of these problems are proven to be convertible between each other. This means that, an algorithm capable of solving one such problem efficiently can be used to solve others with some adjustments.

1.1. Classification of Problems

In computer science, optimization problems are divided into two basic categories based on the complexity of the problem with respect to its input size. Complexity of a problem is the number of necessary steps required with respect to number of inputs for the most efficient algorithm for the problem.

Class P Problems: If there is an algorithm that can solve the problem in at most n^c steps where n is the input size and c is a constant. Since constant c is the most significant factor determining the running time, we call it a polynomial time algorithm and the problem it solves becomes class P optimization problem. Sorting problem can be considered as an example for this class. One of most efficient known algorithm for this problem is Tim Sort which makes $n \log n$ comparisons at worst and average cases. Average number of comparisons made by Tim Sort for different input sizes are given in Table 1.1.

Table 1.1. Tim Sort average number of comparisons depending on instance size

Input Size	Average Comparisons ($n \log n$)
10	10
20	26
30	45
50	85

Unlike Traveling salesman problem, increasing input size does not get overwhelming as the input size increases.

Class NP – Hard: NP (non-deterministic polynomial) problems do not have polynomial time algorithms for their solution. An NP problem is convertible to other problems in NP domain, so that, if there is a polynomial time algorithm found for this problem, all

other problems in NP class can be solved in polynomial time. Since they are convertible between each other, any problem in this class is thought to be as hard as the hardest problems in optimization.

It is currently not known whether NP-Hard class problems can be solved in polynomial time, since $P = NP$ formula has not been proven. However, it is generally believed that $P \neq NP$ (Cormen, 2013).

Table 1.2 shows the number of solution candidates to consider for an algorithm for solving travelling salesman problem with brute force approach. Most efficient exact algorithm for TSP known today takes an average of $O(n^2 2^n)$ steps to complete (Woeginger, 2003). Table 1.3 shows the number of solution candidates to consider for this algorithm.

Table 1.2. Brute force TSP Solution average number of permutations to test depending on instance size

Input Size	Number of Candidates to test ($n!$)
10	3,628,800
20	2,432,902,008,176,640,000
30	265,252,859,812,191,000,000,000,000,000,000

Table 1.3 Most efficient TSP Solution average number of permutations to test depending on instance size

Input Size	Number of Candidates to test $O(n^2 2^n)$
10	102,400
20	419,430,400
30	966,367,641,600
50	2,814,749,767,106,560,000

Even with most efficient exact algorithms, runtime will be greatly increased in case of NP-Complete problems. However, optimal results might not be mandatory in all situations. In such a case, one might weaken the requirement of finding absolute optimum and accept a “good enough” solution. In this case, if the problem has a good known heuristic, they provide good enough solutions in a reasonable time. If the problem in question have no known best practice for its solution, problem-independent metaheuristic methods can be applied to obtain a good enough solution within a reasonable time frame.

1.2. Quadratic Assignment Problem as an NP- Hard Problem

Just like Traveling Salesman problem, Quadratic Assignment Problem (QAP) is one of the most difficult combinatorial optimization problems. Quadratic Assignment Problem was first proposed by Koopmans and Beckmann in 1957 to formulate economical activities. The problem consists of assignment of a number of facilities to the same number of locations in such a way that each location will have a facility. In this problem, distances between locations and flows between the facilities are known. The objective generally considered in the literature is minimizing the cost of allocating facilities into locations, where cost is the sum of all possible distance-flow products.

Since its first proposal by Koopmans and Beckmann, this problem has been used in blackboard-wiring, numerous economic problems, building a decision framework for assigning police stations, supermarkets and schools, scheduling problems, best design for typewriter keyboards and control panels, archeology, statistical analysis, analysis of reaction chemistry, numerical analysis, hospital planning etc. (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007).

QAP is known to be one of most difficult combinatorial optimizations and serve as a benchmark problem for evaluating metaheuristics algorithms. Sahni and Gonzales proved this problem is NP-Hard and in general instance sizes of $n > 30$ cannot be solved by traditional exact algorithm approaches in a reasonable time (Sahni & Gonzales, 1976). That means unless it is proven that $P = NP$, there is no known way to solve this problem in a polynomial time in exact approaches.

Reductions can be made to QAP to transform it into Traveling Salesman Problem, Binary packing problem and maximum clique problem. So, any algorithm capable of solving QAP in a reasonable time interval will also be able to solve these problems

when the necessary transformation algorithm is supplied (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007).

QAPLIB is an Internet library that is created by Stefan Karisch in 1991 which contains QAP instances from literature, as well as, published best known and optimal solutions, latest news about QAP, lower bounds found in literature for the instances, software dedicated to QAP, people who have been working on QAP (Burkard, Karisch, & Rendl, 1997). It was maintained by Karisch until 1997. From 1997 to 2012, Erande Cela maintained the library who is also famous for his work on QAP. From 2012 and onwards website is maintained by Peter Hahn and Miguel Anjos from University of Pennsylvania and Polytechnique de Montreal. Website contains 15 well known data-sets in literature including most famous Nugent, Skorin-Kapov and Taillard datasets ranging from sizes 12 to 256 (Burkard, Karisch, & Rendl, 1997).

Library contains 135 instances from 15 data-sets in total and according to Taillard (Taillard É. , 1995) and Tseng & Liang (Tseng & Liang, 2006) these instances are grouped into 4 categories mainly:

1. **Real-life Instances:** Instances created after real-life problems, these types of problems have many 0 entries on weight matrix and other entries are not uniformly distributed. Burkard and Offermann data set, Steinberg data set and Krarup data set are main examples.
2. **Real-life like Instances:** This kind of instances resemble real-life problems, however, they are artificially created and generally larger on size. Taillard “b” instances are on this category.
3. **Uniform Randomly Generated Instances:** Entries in their distance and flow matrices have been generated using a uniform random distribution. According to Taillard, they are the most difficult ones to solve. (Taillard É. , 1995) Taillard “a” instances are on this category.
4. **Random Flows on a Grid:** Instances in which locations are placed on an $n \times n$ grid and distances between these points are generated as Manhattan Distances between grids. Nugent, Skorin-Kapov and Wilhelm and Ward instances are in this category.

1.3. Metaheuristics for NP Hard Problems

Many of the optimization problems in literature are considered to be NP-Hard and a lot of computation effort is required to solve them. There are approaches like backtracking, enumerative search, branch and bound methods, dynamic programming which can find optimal solutions but they are only useful for specific problems because finding an optimum solution for NP-Hard Problem is often extremely difficult.

However, for some problems, even for some problem instances, there might be shortcuts only applicable to that problem, or problem instances. A heuristic is any kind of solution technique which allows some degree of simplification to the problem at hand, which may or may not give the optimum solution. Heuristics are usually derived from human experiences and allow reasonable speedups, though, as mentioned, they do not guarantee finding optimal solutions. When finding an optimum solution is considered to be hard using conventional ways, researchers usually look for a heuristic which can be applicable to the problem at hand. Most common heuristic techniques are trial and error, educated guessing, intuitive judgements, common sense, etc.

When there is no known heuristic for a problem at hand, researchers look for more generic methods which are applicable to wider area of problems. Metaheuristics are a set of problem independent methodologies designed to be applicable to wide diversity of problems, which often yields good enough, approximate (sub-optimal) solutions. They are extensively used in optimization and machine learning.

There are two primary mechanisms of metaheuristics; exploration, that is, exploring a wide portion of search space and exploitation, which is finding local optimum points as much as possible. A good metaheuristic balances both mechanisms and provides sufficient results in a reasonable amount of time. In literature, metaheuristics are classified under 2 categories: trajectory based metaheuristics and population based metaheuristics.

Trajectory based metaheuristics use a single candidate solution and this solution is modified iteratively as the algorithm progresses; when a better solution is obtained within its neighborhood, original candidate is replaced until there are no more better solutions. Due to this trait, such algorithms create the illusion of following a trajectory within search space. Some examples of trajectory based metaheuristics are simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), tabu search (Glover, 1989) (Glover,

1990), iterated local search (Lourenço, Martin, & Stützle, 2003), variable neighborhood search (Hansen & Mladenovic, 2001) and greedy randomized adaptive search procedure (Feo & Resende, 1995).

Second type of metaheuristics favor a collection of individuals over a single individual, namely called as population based heuristics. As such algorithms progress, individuals within the population interact with each other and create varieties. In time, unfit solutions are neglected, usually superior solutions are selected, combined to create better solutions. This process continues until a reasonable solution is found. Examples of such algorithms are evolutionary algorithms (Eiben & Smith, 2003), particle swarm optimization (Kennedy, 2010) and ant colony optimization (Dorigo, Birattari, & Stützle, 2006), scatter search (Glover, Laguna, & Marti, 2003), etc.

Trajectory based problems metaheuristics are generally better on exploitation, they find results faster and are more computation power efficient but they are more prone to being stuck at local optimum points since diversity is a lesser concern. On the other hand, population based metaheuristics focus more on exploration of search space and thus, are slower but, generally they are more robust and less prone to being stuck at a local optimum. However, we cannot infer one is better than the other in general.

Besides traditional metaheuristic methods, there are also hybrid models which combine several methods to solve optimization problems. A common approach is having a population based metaheuristic as the base problem solver and improving it with a trajectory based method, thus utilizing stronger sides of each approach. Another possible improvement for such methods is utilizing parallelization and parallel programming in order to run multiple instances of metaheuristics algorithms to improve overall search result quality and reduce the runtime of algorithm.

1.4. No Free Lunch Theorem

A theorem presented by two computer scientists, David Wolpert and William G. Macready in 1997, states that, there cannot exist an algorithm which solves all optimization problems better than other competitor algorithms (Wolpert & Macready, 1997).

This theorem is widely accepted today, because no one was able to come up with such an algorithm. So, for instance, comparing evolutionary algorithms, in broader context,

evolutionary computation methods with other types of computation methods and algorithms in general is meaningless. More sensible approach would be comparing evolutionary algorithms and other type of problem solvers in specific problem types.

According to a survey made by Hans-Paul Schwefel, an EA can perform well when solving discontinuous, non-differentiable, multimodal, noisy and otherwise, when unconventional response surfaces are involved, while it is weaker in solving linear, quadratic, strongly convex, unimodal, separable types of problems (Schwefel, 1997).

NFL theorem tells us that there will always be a trade-off between reliability of problem solver and its applicability to problems in general. When solving a problem, if there is an efficient algorithm which can solve problem in question, metaheuristics should not be utilized. However, metaheuristics are solvers which are easily applicable and easily adoptable for a wide variety of problems. One other quality they provide is their robustness; they are not easily affected by parameter changes in the problem. In short, metaheuristics are a common ground between variety of problems and applicability of problem solvers.

1.5. Genetic Algorithms as Metaheuristics

Genetic algorithms are the most common subtype of evolutionary algorithms. They are considered as population based metaheuristics. Evolutionary computing is a major area of research in combinatorial optimization which draws inspiration from natural selection and evolution of species as well as survival of the fittest. Conception of evolutionary computing is no surprise, because nature has proven itself to be a formidable competition environment, a source of inspiration for a problem solver. That is, any species in nature that has tailored itself to its environment or niche over a long time. Just like nature, evolutionary computation draws its strength from Darwin's "Survival of Fittest" principle. And just as in nature, evolutionary algorithms start with a population. This population will produce new individuals over time through breeding (cross-over), mutation and selection of stronger. In this setting, over many generations, stronger traits will overtake the population, while weaker traits will disappear over time. When applied to optimization, an individual solution's fitness value determines how well it will adapt itself inside the population and become a source for the later generations. This process yields better and better results for a certain amount of time.

In the end, usually fair-enough solutions are obtained when the literature is overviewed. Genetic algorithms are no exception.

1.5.1. GA Properties

Genetic Algorithms subclass is first proposed by John Holland in his book *Adaptation in Natural and Artificial Systems* in 1975 (Holland, 1975). According to Eshelman, there are three distinct specifications which separates genetic algorithms from other types of evolutionary algorithms. These are representation used (bitstrings), selection method (proportional selection) and primary means of obtaining newer solutions (cross-over). However, over the years, new representation methods such as real-digit values, permutations, floating points and other selection techniques like stochastic universal sampling, rank based selection, tournament selection etc. are adopted in distinct implementations. This made Genetic Algorithms closer to other types of evolutionary algorithms like Evolutionary Strategy, Genetic Programming, Evolutionary Programming, while primary emphasis on cross over technique always remained in later implementations (Eshelman, 1997).

In his book, Holland created the first genetic algorithm which will be latter known as “Simple GA” or “Canonical GA” to optimize mathematical functions. Over time, genetic algorithms gained different representations and selection methodologies, in addition, high mutation rates are used in different studies. Table 1.4 lists the general properties used in Simple GA.

Table 1.4 Simple GA Properties

Representation Method	Bit Strings
Recombination	Single – Point Cross over
Mutation	Bit flip
Parent Selection	Fitness Proportional
Survivor Selection	Generational

GA General Approaches

In all genetic algorithm implementations, problem solving process starts with a population which consists of a number of usually randomly created individuals. Representation of these individuals are called chromosomes and they can be an integer number, a bitstring, a floating number or permutations. And then, each of these individuals are evaluated based on a predetermined function to evaluate their fitness values. Those fitness values are called phenotypes and they determine the chance of selection of each individual for mating (cross-over). In the next step of algorithm, some of these individuals are selected from the population and cross-over operator is applied. Since their chance of selection is based on their fitness values, there is a greater chance the better individuals inside the population are selected and in rare occasions, the weaker can be selected to provide diversity within the population. When cross over and usually low amount of mutation is applied, those selected individuals create a number of new individuals referred as offspring or children. Each of the new offspring are evaluated by the fitness function and their fitness values are also determined. After new offspring are created, algorithm will have a parent population and child population. Since most of the genetic algorithms prefer fixed-size populations, a number of individuals must be selected from two populations to create next generation. To do that, a selection operator is applied. Selection operator may prefer replacement of all parents in the population with children, a random selection of n individuals or selection based on fitness. This part of algorithm ends with a new generation of individuals. The same procedure applied to newly obtained generation and the ones come after it, until one of these newer generations satisfy the termination condition. This termination condition can be number of generations, finding a predefined fitness value, absolute time passed or total number of fitness calculations, etc. When a viable solution is found, the algorithm terminates. A sample pseudo code for GA is as follows:

```

BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END

```

Figure 1.1. A Pseudo – code for GA (Eiben & Smith, 2003)

General processes in an evolutionary algorithm can also be represented in a flow-chart diagram as shown below:

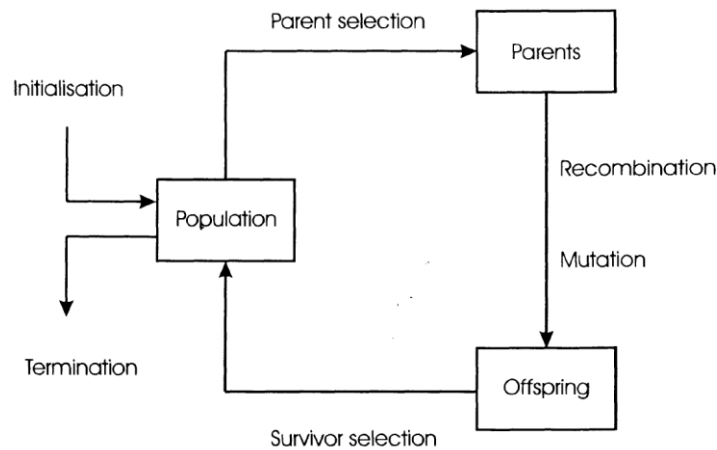


Figure 1.2. An Iteration Cycle of GA (Eiben & Smith, 2003)

As described above, genetic algorithms work through the iterations of the loop, what is called generations. First branching starts on survivor selection. In classical GA, algorithm creates a number of children from randomly selected parents and once this number is reached, it replaces all the parents with newly produced children, thus creating the next generation. This kind of genetic algorithms are called Generational GA. Other kinds of implementations may favor combining parent and child populations and selecting n individuals from merged population. This kind of genetic algorithms are called Steady State GA.

In a Steady State GA, in addition to crossover and mutation, a replacement operator is also required. This kind of algorithm make replacements without regarding the generation or iteration count. Newly produced children are introduced to population

as soon as they are created. This approach allows user to precise adjustment of the course of algorithm, while a Generational GA, this work will be done by selection method which will be used to determine parents. This makes implementation of Generational GA simpler to implement but harder to control its course. Additionally, a Steady State GA will guarantee good individuals will be kept for next generation while a generational GA will need a property called elitism to have same feature.

GA Selection Methods

In each iteration of the algorithm, a subset of the current population is selected to breed new individuals. This is the main process that drives algorithm further. This selection of subset is usually based on a function called “Fitness Function” which determines the quality of the individuals in population by assigning them positive score values. Typically, as the score gets higher, it will increase the chance of selection of the individual.

A selection procedure would start with evaluation of each of the individuals inside the population according to fitness function. This fitness function is always problem specific; meaning that, each problem has its own fitness calculation. In some problems, additional constraints might be given like time and budget. So, fitness function must also account satisfiability of these constraints. There are different approaches to ensure validity of solutions. The algorithm might simply discard invalid solutions, it might attempt to repair invalid solutions, may prefer to use task-specific operators that always produce correct solutions or may integrate a penalty mechanism for invalid solutions. In the next step, population is generally sorted in an ascending order with respect to the fitness scores of individuals. After that, selections are made according to the properties of various algorithms. Most commonly used algorithms for selection are; Fitness Proportionate, Stochastic Universal Sampling, Rank Selection, Tournament Selection and Truncation Selection, while in some cases selection can be made randomly to gain from computation time.

- a. Random Selection: The weakest and most obvious selection type is simply picking n number of individuals from population for breeding. As the selection will be totally random, it will not help the forward development of algorithm results. In most cases this approach is used in steady state algorithms to gain computation time where selection pressure is enforced by replacement strategy.

- b. Fitness Proportional Selection: This is the most straightforward selection approach and is utilized by simple GA. Entire population is sorted based on fitness quality of individuals, and then, a probability value is created for each of the individuals by normalizing fitness values. In each iteration, one individual is selected from the population based on their probabilities. The chance of selection is increased according to the fitness value; so fitter solutions are more prone to being selected. The algorithm takes n iterations for n selections and therefore it will have $O(n)$ running time. The advantage of this method is that, it is the simplest conventional method, while, its disadvantage is that, it takes significant computation time and a super fit individual may take over the population in a short time.
- c. Stochastic Universal Sampling: It is a more advanced form of fitness proportional selection introduced by James Baker (Baker, 1987) . Unlike FPS (Fitness Proportional Selection) algorithm, which selects a number of solutions from population by repeated sampling, this algorithm divides the population into evenly spaced ordered based on fitness intervals. SUS then rolls a random number to sample solutions from each of the intervals. This algorithm has advantages over classical FPS algorithm: it increases chance of weaker members of population to be selected, thus, increases diversity within population. It is considered a fairer approach than classical FPS.
- d. Rank Selection: While fitness proportional and SUS methods are strong, they suffer from their sensitivity to relative individual fitness values. For instance, a solution with fitness value 10 is 10 times likely to be selected than a solution with fitness value 1. On the other hand, an individual with fitness value 110 has almost the same chance of being selected with an individual with fitness value 100. In order to solve this weakness, rank selection algorithm's selection depends on the relative strength of solutions.
- e. Tournament Selection: Basic idea of this algorithm is creating a number of tournaments between several solutions which are selected randomly from the population. Winner of each tournament is then selected for breeding. In addition, the winner is determined by a probability value which allows higher chance for better individual and allows a lower chance for worse individuals. If the tournament size is adjusted, it allows user to influence selection pressure. Higher tournament size creates more selection pressure. This algorithm works well in

parallel processing, is simple to code and is as efficient as other fitness proportionate algorithms and allows the adjustment of selection pressure.

- f. Truncation Selection: In this model, entire population is ordered based on fitness values and a subset of fittest individuals are selected for breeding. While this method is less complex than the others, it is not used in general.

Elitism is usually used in generational algorithms in order to retain the best individuals inside the population. To put it simply, most quality solutions within the population are preserved and passed on to the next generation. This approach is not required in a steady state algorithm, since it relies on replacement strategies to retain the best individuals.

GA Replacement Methodologies

Replacement also known as survivor selection is a key component in determining algorithms progress. In generational GAs, this selection is simply based on selecting newly produced children and eliminating parents from the new population, while in steady state GAs, there are different approaches and each has their own strengths and weaknesses:

- a. Random Replacement: The simplest and most straightforward approach for steady state algorithms is generating new population by selecting individuals from both parents and newly created children randomly. If selected, this approach will not drive algorithms search towards a specific course and leave the driving search to better individuals to selection method. When used in a steady state GA, this will bring it close to traditional generational GA, where the progress of the search is also dependent on selection method. While it has the advantage of easy implementation, it will have no significant difference from a generational GA.
- b. Inverse Proportional Replacement: As the name suggests, this methodology is somewhat similar to fitness proportional selection used in parent selection. Probability of being replaced in the next generation is determined by a probability that is inverse proportional to individual's fitness quality. This means that, the lower fitness value of individual, the higher chance it should be replaced. This, of course, guides the search process to better individuals very fast, but, after an extended period in algorithm's course, all population members will have similar

characteristics and therefore similar fitness values. This may cripple algorithm's diversity and prevent it from enhancing solution quality.

- c. Replace Worst: In a steady state GA, most commonly used method is replacing the worst fitness quality members of the population with the newly created children. This approach would also drive the search process further, eliminating weaker candidates in a time interval and creating more robust population in a relatively shorter time frame and preserving the best candidates is ensured. When combined with a random selection strategy, it may yield satisfactory results. However, in a long period of run, this approach is thought to kill the diversity within the population and all candidates will become somewhat similar, and finding good results will be significantly harder and it may have considered to be a greedy approach.
- d. Replace Parents: This idea is based on newly created children to replace their worst parent or all of their parents. While the clear advantage of this is always maintaining some form of diversity within the population, it will not speed up the search process as much as replace worst strategy. It is slower, but it may not be effected by the degradation of diversity as much.

GA Representation Methods

Regardless of which type of GA is used, strategy of building a genetic algorithm always starts with deciding on a proper representation. Without an efficient representation, it might be difficult to differentiate the quality of individual solutions. Choosing the right representation is one of the most difficult aspects in genetic algorithm because everything else, selection strategy, replacement strategy, choice of crossover and mutation operators will depend on it. In order to do the right choice, a good knowledge about the problem to be solved must be present, while, some common choices used in literature might be helpful for this decision.

- a. Binary String Representation: A solution is represented in the form of a string of binary digits. To use this representation, size of the string and technique to convert genotype consisting of binary string to a suitable phenotype must be decided upon. If a binary string of a particular size can encompass entire search space, this kind of a representation is efficient. If the problem has extra constraints which creates invalid solutions, such type of solutions must be fixed or dealt with.

In literature, binary string representations is used for many problems such as knapsack problem or one-max problem.

- b. Integer Representation: If the problem to be solved contains a set of values which can take integer values as solution, interpretation with binary representation might slow down the process or might not encompass entire search space. In such cases, representing solutions using integer numbers might be more practical.
- c. Floating Point Presentation: There are also cases when set of candidate solutions are continuous rather than discrete. In such problems, although not entirely accurate, most suitable form of presentation is using floating point numbers.
- d. Permutation Presentation: When a problem needs to be presented as a sequence of events, this kind of solution representation is often preferred. Events can be labeled with integer values and their sequencing within the permutation might present the order which events occur.

As examples, generally algorithms solving 8-Queen problem, traveling salesman problem, quadratic assignment problem prefers this kind of representation.

GA Mutation Methods

A mutation procedure is introduced into genetic algorithms to generate diversity within the population as the algorithm progress. Unlike an Evolutionary Strategy algorithm, where mutation operator is used to direct the search process, in genetic algorithms mutation is used strictly to create variations and widen the area covered by the algorithm's search process within the search space. Technique of mutation is heavily dependent on genetic structure of solutions (genotype). There are different types of mutation operators used in literature:

- a. Bit-Flip Mutation: This technique is used in binary string representations. Given a small probability, a candidate solution can undergo a change in one of values in its bits. If the bit is 0, it is inverted to 1, if it is 1, it is inverted to 0.
- b. Random Resetting: Used in integer representations where an integer value is replaced by one of allowable values in problem domain with a small probability.
- c. Creep Mutation: Used in integer representations by adding a small value to the genotype by a small probability. Probabilities are computed for each position within the genome. Generating small changes is more probable than larger changes.

- d. Uniform Mutation: Works on floating point representations with the principle of increasing or decreasing an allele value in the range of a value randomly selected within a number domain.
- e. Swap Mutation: Works on permutation representations by swapping the allele values of two randomly selected genes (positions). An example is shown in Figure 1.3.



Figure 1.3. Sample swap mutation between two alleles

- f. Insert Mutation: Used in permutation representations. In this method, two points are selected at random. One of the points is moved to the end of the other. An example is shown in Figure 1.4.



Figure 1.4. Insert mutation between two alleles

- g. Scramble Mutation: Used in permutation representations. Two points are selected to create a subset and then this subset is shuffled. An example is shown in Figure 1.5.



Figure 1.5. Scramble mutation

- h. Inversion Mutation: Inversion works by the principle of selecting two points and creating a subset and inverting its order. An example is shown in Figure 1.6.



Figure 1.6. Inversion mutation

GA Recombination Methods

Recombination can be thought as creating newer individuals by combining the genes of parent individuals in a meaningful way, without violating the problem constraints. It is one of the most important aspects of a genetic algorithm. While mutation is mainly used for creating diversity within the population by occasionally diverting some solutions from the local maximum points, recombination (cross-over) operators are mainly responsible for driving the search process further by pulling the population into the local maximum points. To make search process efficient, two operators should be used in balance. Just like mutation methods, cross-over methods are also dependent on the problem representation.

- a. Single-Point Crossover: A single point is selected in both parents' chromosomes. And then, parents' chromosomes are splitted at this point. Two children are created by exchanging tail parts. Used in binary representations and integer representations. An example is shown in Figure 1.7.

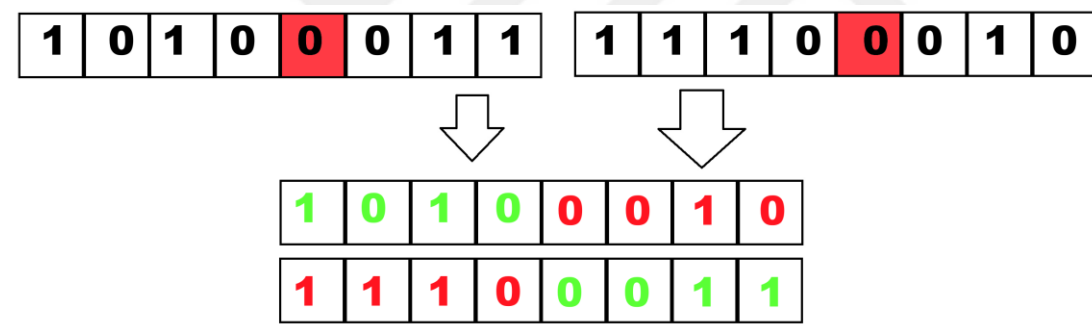


Figure 1.7. Single Point Crossover

- b. N Point Crossover: N Points are selected within the chromosomes of parent individuals and divided. And then those different segments are merged in children. Used in binary representations and integer representations.
- c. Uniform Crossover: Instead of taking full chromosomes of parents, this method works on genes. For each gene, a probability value is created and based on that value, source of the gene whether parent 1 or parent 2 is selected. Works in binary representations and integer representations.
- d. Single Arithmetic Recombination: Picks a gene on a random position in both parents and take the arithmetic mean of the parents' allele at that point and further

beyond that point. Takes head section directly from parents. Used in floating point representations.

- e. Whole Arithmetic Recombination: Works by taking arithmetic mean of two parents' allele on all genes. Two created children have identical chromosomes. Used in floating point representations.
- f. Partially Mapped Crossover (PMX): Used in permutation representations. First introduced by Goldberg and Lingle as a cross-over operator for the solution of TSP (Goldberg & Lingle, 1985).

The algorithm first selects two points within parent chromosomes randomly and copies the genes between these two points from first parent. Then checks the alleles in the same section of second parent that are not copied into child. For each of these alleles (say A), checks the alleles in their positions on parent 1 (Say B). And then, places A in child to the position of B in parent 2. If the place of B in child is occupied by the element C, copies the allele A into position of C in child. And at the end, copies the rest of the alleles from parent 2 into child. And then the second child is created by parents 1 and 2 reversed.

A simple illustration of how the process works is given in Figure 1.8. Whole process takes 3 phases. In the first phase, alleles from parent 1 are copied directly to child. In the second phase, alleles not present in child but present in parent 2 are copied. In the last phase, all other alleles are copied into their original positions in child.

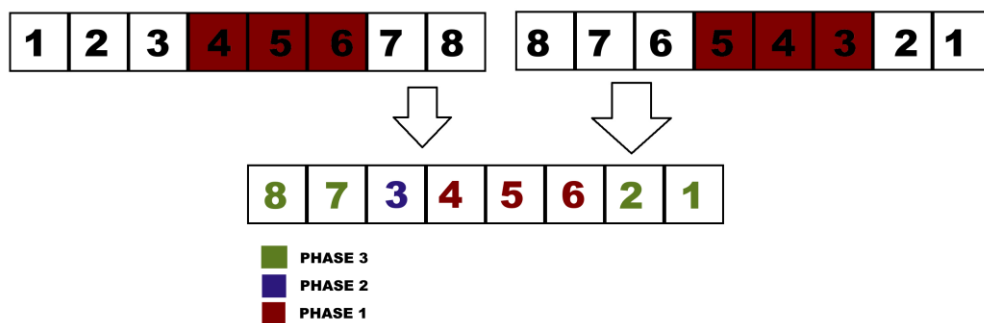


Figure 1.8. Partially Mapped Crossover

- g. Edge Crossover: It is another interesting idea for permutation representations that favors the preservation of edges found in both parents. In this algorithm, first, neighboring alleles for each allele is determined and they are listed. In the second step of the algorithm, a random allele is selected and in each iteration,

algorithm might select one of the edges. Selection firstly favors edges found in both parents. If no such edge is present, next allele is selected from one of the selectable edges randomly. An example is shown in Figure 1.9.

Parent 1:	1	2	3	4	5	6	7	8
Parent 2:	4	7	1	5	3	8	2	6

Element	Edges	Selection	Next Choices	Children
1	8-2-7-5	3	2458	3
2	1-3-8-6	2	186	32
3	2-4-5-8	8	71	328
4	3-5-6-7	1	75	3281
5	4-6-1-3	5	46	32815
6	5-7-2-4	4	67	328154
7	6-8-4-1	7	6	3281547
8	7-1-3-2	6	--	32815476

Figure 1.9. Edge Crossover

- h. List Order Crossover: This is another crossover technique derived from PMX. It starts in the same manner as PMX: a subset of genes are selected from parent 1 and are copied into the child. In the next step, starting from the endpoint of randomly determined subset, the unused alleles in parent 2 are copied into the children in the same order in parent 2, until no more element remains. A simple representation can be shown as below in Figure 1.10:

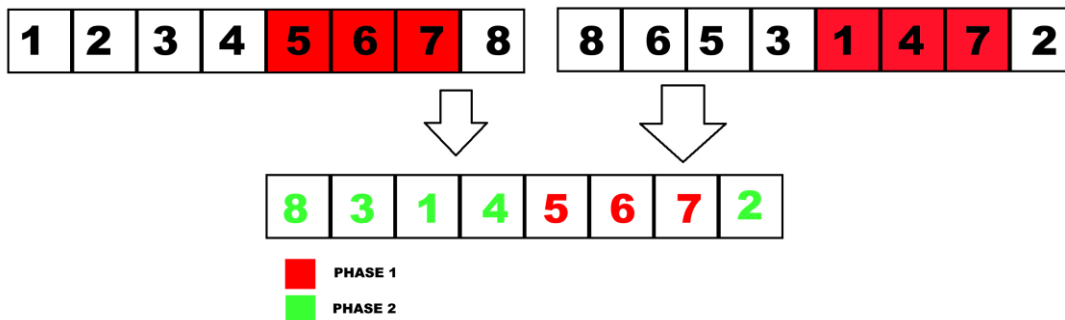


Figure 1.10. List Order Crossover

1.6. Parallel Computing

Originally, all of the programs that run on a computer were designed to run in serial. They were split into sets of instructions which are sent to a single processing unit and executed in sequence one by one. This resembles an analogy where a single worker is building a wall, doing all necessary tasks one at a time. This approach was slow and

complex tasks requiring lots of computing power would require very long-time frame to complete (Figure 1.11).

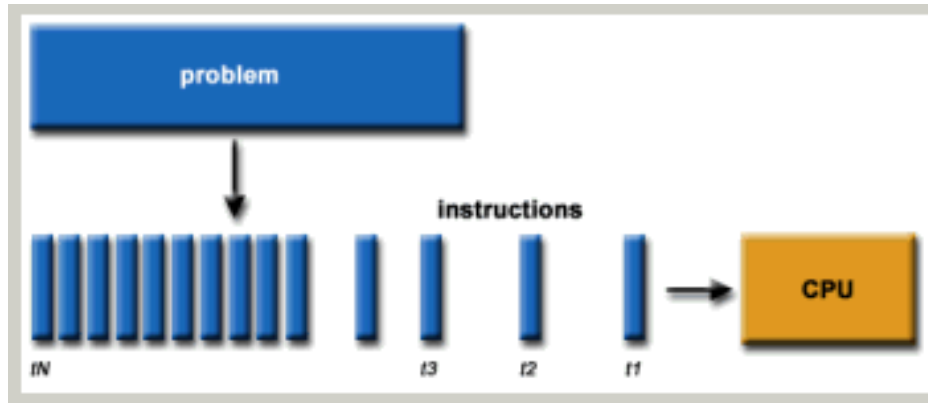


Figure 1.11. Serial Processing (Barney, 2009)

In contrast, parallel programming is the use of multiple processing units simultaneously to by dividing a task at hand into smaller chunks (Figure 1.12).

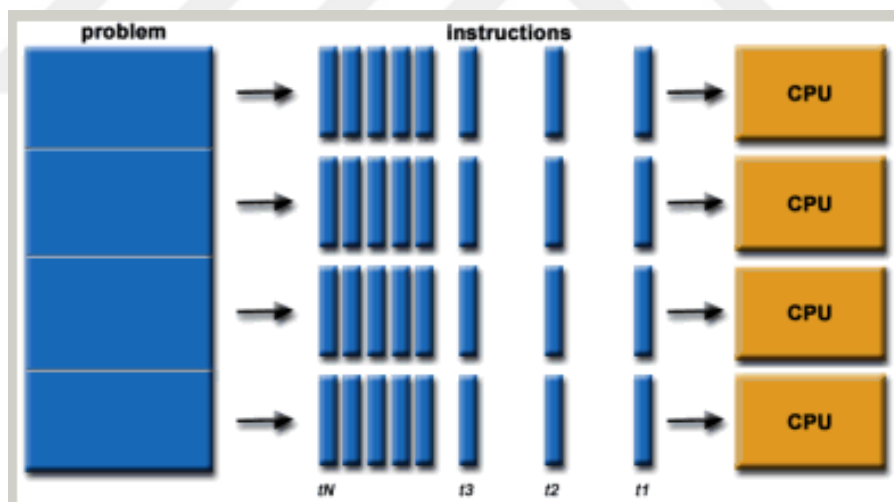


Figure 1.12. Parallel Processing (Barney, 2009)

This processing unit maybe multiple cores of a single processor, multiple processors on a single computer or a cluster of computers. In order to achieve an efficient parallelization, problem at hand must be logically divisible into smaller subproblems. Today, parallel software systems are used in numerous fields effectively, such as, big data processing, data mining, search engines, medical imaging, computer graphics,

virtual reality, web services, entertainment industry, operating systems, computer security as well as optimization.

Where it is possible to use multiprocessing, it yields huge benefits. It decreases the runtime required for solving problems, allows to solve larger instances of the problem and since all the processors today contain several processing units, it also allows efficient utilization of resources at hand.

1.6.1. Flynn's Taxonomy

Proposed by famous computer researcher Michael J. Flynn in 1966, Flynn's taxonomy is a classification of parallel computing architectures based on instructions and data (Flynn, 1972). It is still most widely accepted and is a popular classification technique as of today. There are four distinct categories in this classification.

- a. Single Instruction Single Data (SISD): A serial computer which features no parallelism. A single processing unit fetches a single instruction from memory at a time and only a single data stream is used as an input at a time. Typical examples are old computers and mainframes.
- b. Single Instruction Multiple Data (SIMD): A type of parallel computer in which all processing units fetch and execute same instruction from memory at a time where each of which uses different data streams as inputs at a time. A typical example can be GPU unit.
- c. Multiple Instructions Multiple Data (MIMD): Most common type of parallel computers today, in which multiple processing units fetch and execute different instructions from memory and are able to use different data streams of their own. Typical examples are modern personal computers, tablets and smartphones.
- d. Multiple Instruction Single Data (MISD): An uncommon type of parallel computer in which each processing unit fetches different instructions from memory and operates on same data stream.

1.6.2. Parallel Computing Performance Criteria

Performance gain in multiprocessor architectures are often calculated by Amdahl's law. According to this law, performance gain by enhancing some portion of a computer can be calculated using "Speedup Ratio" formula (Amdahl, 1967).

- a. Speed-up Ratio: A ratio which determines how much gain will be obtained when an enhancement is made. Can be adapted to multiprocessor architectures. Therefore, speed-up ratio on N-Cores (S_N) can be calculated as follows:

$$S_N = \frac{\textit{Execution Time for Entire Task in Single Core}}{\textit{Execution Time for Entire task in N – Cores}}$$

- b. Parallelization Efficiency: Determines how much speedup is obtained per newly added processor:

$$E_N = \frac{\textit{Speed up Ratio } (S_N)}{\textit{Number of Cores utilized } (N)}$$

There are three possible scenarios for parallelization efficiency when a program is divided into N processors:

$$E_N = \begin{cases} \textit{Sub – linear efficiency} , & E_N < 1 \\ \textit{Linear efficiency} , & E_N = 1 \\ \textit{Super – Linear efficiency} , & E_N > 1 \end{cases}$$

In most cases, increasing the number of processor cores will not yield the efficiency desired. The biggest reason for that is, it is not possible to divide all of the portions of the program N-Equal tasks. That is why most of the time, sub-linear efficiencies are acceptable. However, in some cases, linear efficiency can be obtained. Additionally, execution time may not be the only gain in multi-core architectures. There may be other performance considerations depending on the problem.

CHAPTER 2

PARALLEL EVOLUTIONARY ALGORITHMS

Genetic algorithms in general are types of metaheuristics that provide good enough results within a reasonable time frame for computationally hard problems. But when applied to larger instances of hard problems, they may become relatively slow. In addition, it is not just a matter of speed but also quality of solutions may drop. Main reason of the drop in the solution quality is that algorithm will only be able to focus a smaller portion of search space, since the expanse of traversing search space will also increase as input size increases. A single sequential algorithm will eventually converge on a small portion of search space and lose its diversity. Since a good metaheuristic is expected to balance solution exploitation and solution exploration, they may become inefficient.

One way to deal with these problems is taking advantage of modern computation architectures and multiple processor cores within a single chip or to put it shortly, combining evolutionary algorithms with parallel computation. By sharing the workload evenly among processor cores, it is expected to gain a speedup over the same algorithm running on single processor core, as well as, running different algorithms simultaneously to discover a larger area in search space with the hope that it will lead to finding better solutions. It should also be pointed out that, parallelized EA models resemble natural evolution process better than their serial counterparts.

As genetic algorithms are perhaps the most popular subtype of evolutionary algorithms, parallelization of GA, is also more popular compared to the other types in the literature. There are several strategies suggested in the literature as to how to accomplish parallelization of GA.

2.1. Parallel GA Models

2.1.1. Parallel Independent Runs

A simple approach based on the idea of a single GA might not be able to explore a wide area within the search space. So, by taking advantage of the multi-core architecture of modern processors, several independent GA might run simultaneously. Problem here is; there is no communication between these GA programs. This can be

achieved easily by running each of the evolutionary algorithms on separate threads or processes. Although a simple mechanism, it can be extremely helpful if we are not sure about the nature of problem at hand. By running GA with different parameters (crossover rate, mutation rate, selection strategy, etc.), it allows gathering statistical data about the problem at hand in a much shorter time frame compared to running each configuration sequentially. Disadvantage of this strategy is; since there is no communication between independent runs, diversity within each algorithm will rapidly fall compared to other methodologies and it will conclude on a local optimum with much greater probability. This problem is also known in literature as premature convergence.

2.1.2. Master-Slave Model

This model is constructed to gain pure speedup compared to a simple sequential GA. The idea is that, time consuming tasks like fitness evaluation, crossover and mutation within a GA can be distributed to the other processor cores and completion time of the algorithm can be reduced drastically. Main parts of the algorithm like initialization and generation loop will be dealt within the master process/thread, while time consuming tasks, mostly fitness evaluations, will be distributed to slave threads/processes assigned on different cores. Once the calculations are complete, slave threads are terminated and results are returned to master thread. The other parts of the algorithm will work just as its sequential counterpart. The experiments carried out in this thesis study show that, when applied to the sequential algorithms, this method yields a significant speedup. Figure 2.1 shows the general architecture of master-slave model.

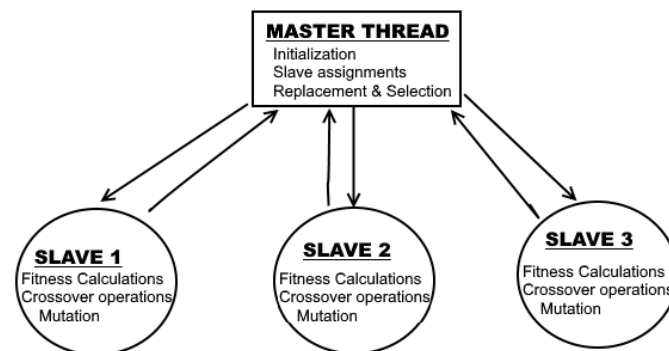


Figure 2.13. Master-Slave Model

2.1.3. Island Model

Another model called island model is an enhancement to independent runs by allowing a communication between GAs on different processor cores. This is expected to eliminate the problem of premature convergence on different GAs and provide better results.

In Island model, also known as coarse-grained GA, main population is divided into different subpopulations known as the islands or demes. Those islands are then assigned into different processor cores and just as in sequential GA, each of them starts to evolve independently. After certain number of generations, known as epoch, each of these island exchange solutions and continue to evolve from that point. Newly introduced individuals add diversity to islands and hopefully keep them from converging prematurely. Also, islands stuck at low fitness regions of subspace are introduced with better solutions allowing them to evolve and contribute.

This model provides a reasonable speed-up. It is also flexible, which means it can be combined with any other model. In addition, it eliminates premature convergence, explores different regions within search space and provides much better results than a classic sequential GA. However, it will add extra parameters due to migration model it implements. These parameters are:

1. **Number of Islands** to make up general population. This determines how many simultaneous runs will be conducted within the algorithm. It is expected that, in general as the number of islands increase, so will the performance of algorithm. But if the general population is divided into too small subpopulations, algorithm may perform poorly. This is usually dependent on problem size and called “critical mass”.
2. **Period of Migration**, which is how often the solutions will be exchanged between islands. If the period is too frequent, subpopulations may not have enough time to evolve and exploit, so algorithm may perform poorly. If the period is too seldom, there is a risk of premature convergence of subpopulations, and therefore, the algorithm may perform poorly.
3. **Number of Solutions to Exchange** within each epoch. If too many candidates are exchanged, newly added solutions might take over the subpopulations and those subpopulations maybe diverted from local optimums. If the candidates to be

exchanged are too few, they may not be enough to prevent premature convergence of subpopulations.

- 4. Immigration Topology**, which determines the spatial distribution of the islands. This, determines the neighborhood structure. Islands can be arranged in a circular order, one after another, in which case immigration topology is called ring migration. If the islands are arranged to resemble a graph migration topology, it is called graph migration. One might also choose to use a multidimensional (3D, 4D, etc.) representation for topology, in which case islands will have more than one neighbors, such representations are called hypercube. Different types of topologies are shown in Figure 2.2.

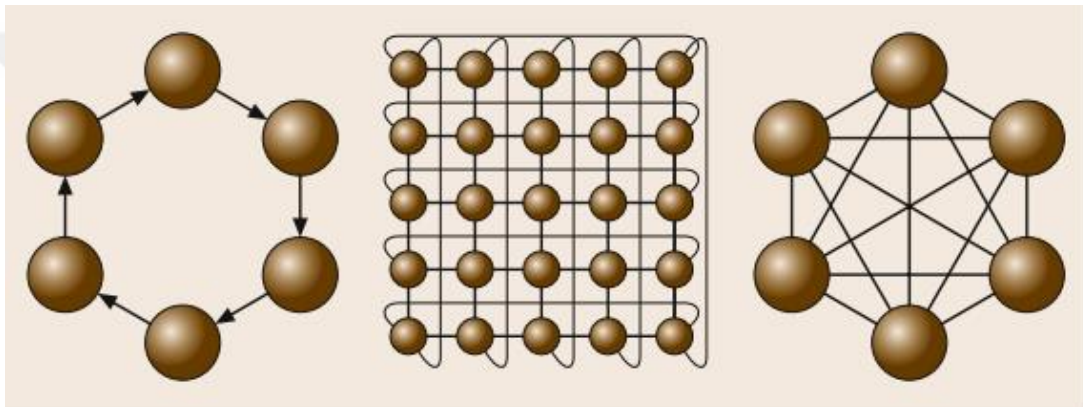


Figure 2.14. A unidirectional ring, a torus and a complete graph (Sudholt, 2015)

- 5. Direction of Migration** between neighbors. Any type of migration topology can be arranged to exchange individuals in a single direction or a bidirectional.
- 6. Immigration Policy**; the decision of which members to exchange between subpopulations. Selection may be the random, the worst candidates or the best candidates or any other selection strategy might also be applied. There is no consensus to which strategy is better.
- 7. Decision to Copy or Move the Immigrants** is the decision of leaving a copy of individual or removing it from original island might also play a part in efficiency of the algorithm.
- 8. Homogeneous or Heterogeneous Island Algorithms.** If all islands are initialized with same parameters, this kind of an island model is said to be homogeneous. If

the configurations of the islands are different, this kind of a model is called heterogeneous island model (Sudholt, 2015).

Island model is ideal for MIMD machines, where every island will be assigned to a different core, therefore this model is commonly favored when a modern-day CPU will be used. But they can also run on single core CPU, albeit not simultaneously.

2.1.4. Cellular Model

This can be considered as a special form of island model, where each island consists of a single individual. It is also known as fine-grained model. Each island is called a cell and these cells are connected with their immediate neighbors in a topology and they can only reproduce or mate with their neighboring cells. As the algorithm progress, neighboring cells will start to be closely related, while, spatially distant cells will be different from each other. This typical change is called isolation by the distance. In the long run, better solutions will start to take over population on a slow rate. This process is called diffusion.

Unlike island model, cellular model implements no evolution within the cells, but it occurs in the outer layer, inter-cells. Commonly, fitness evaluations, selection and mutations and cross over between cells are done in synchronization. However, it is also possible that the algorithms may choose to perform these operations asynchronously in some cases, depending on the problem instance.

An important aspect in this model is the selection of the cell to be updated or order of the updates within the model. Alba et al. (Alba, Giacobini, Tomassini, & Romero, 2002) suggested 4 strategies for choosing the next candidate to be updated:

- Uniform Sweep: Selection is done randomly.
- Fixed Line Sweep: Cells are updated by line.
- Fixed Random Sweep: A sequence of cells determined according to some strategy and updates are done in permutation sequence
- New Random Sweep: A permutation is regenerated after every selection, this model is called new random sweep.

Another important aspect of cellular model is the topology to be selected in which cells will be spatially distributed. According to (Sudholt, 2015) most common topologies

are ring, 2D torus and graphs. This factor will have a large impact on diffusion of better solutions and a correct selection will vastly improve output of the algorithm. Figure 2.3 shows an example of graph representation.

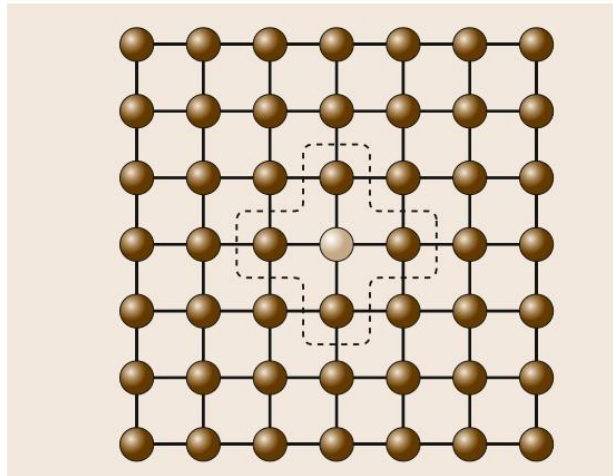


Figure 2.15. Graph representation of the cellular model. (Sudholt, 2015)

Cellular model is preferred on SIMD machines. For this reason, they are very convenient to run on GPU modules. However, they can be modified to run on MIMD standard CPUs we use today, or even on a single core processor albeit not simultaneously.

2.1.5. Hybrid Models

Aside from the traditional models, genetic algorithms can be set to run in combination of these approaches to obtain better speed up and solution quality. However, their implementations will be reasonably more complex. Instead of a single level, there will be two levels of algorithms. Below are some interesting combinations:

1. Island Model & Cellular Model Hybrid: Using an island model on higher level and running a cellular model within each island. Cellular parts may be set to run on computer's GPU, while fitness calculations, migration and other operations are performed on CPU.
2. Island Model & Sub-Island Model Hybrid: Island model on higher level and within each island, running another island model. This may further increase general diversity and allow algorithm to cover search space more extensively.

3. Island Model & Master-Slave Model Hybrid: This model is beneficial where there are more cores in CPU than number of islands constructed on the algorithm to better utilize the available processor.
4. Cellular model & Master-Slave Model Hybrid: In this model, fitness calculations will be sent to CPU for faster computation, and other parts of algorithm will remain on GPU.

2.2. Measuring Performance in Parallel GAs

There are two main factors to assess the gain of parallelization in genetic algorithms. These are computation time gain and solution quality.

2.2.1. Computation Time Gain

Computation time in genetic algorithms is measured in several different ways: wall clock time it takes to complete the algorithm and number of generations it takes to complete the algorithm. In either case, computation time gain can be measured in terms of speed-up method formulated by Ahmdahl (Amdahl, 1967).

In 2002, Alba presented a taxonomy in which speedup provided by the parallelization of evolutionary algorithms can be classified (Alba, 2002):

- a. Strong speedup: Total runtime of the parallel evolutionary algorithm is compared to the runtime of the best-known sequential algorithm. This sequential algorithm may or may not be an EA. This scale determines how much a parallel EA improves over current best-known counterpart. However, it is not favored by the researchers because of the complexity of finding the best known sequential algorithm.
- b. Weak speedup: Total runtime of the parallel evolutionary algorithm is compared to its sequential counterpart. It is favored by the researchers since it is easy to classify. It has two different sub versions:
 - i. Panmictic (Single Population) weak speedup: Runtime of parallel evolutionary algorithm is compared to the single-population version of it on an m processor system.
 - ii. Orthodox weak speedup: Runtime of parallel evolutionary algorithm on a multicore machine is compared to runtime of sequential algorithm on a single core machine.

According to Alba (Alba, 2002), parallel evolutionary algorithms can not only achieve a linear speedup, but also super linear speedup is also possible on some instances. Alba concludes that several factors such as using efficient data structures, better exploration of search space from multiple positions or increase on some other resources like memory, cache etc. effect the efficiency of the algorithm. Therefore, he concludes that achieving super linear speeds in an EA is possible with a correct implementation.

2.2.2. Solution Quality Gain

A parallel algorithm will evidently better discover the different portions of subspace than its sequential counterpart and avoid problems like premature convergence. Effectively this will improve the solution quality.

A formulation called **Relative Percentage Change (RPC)** can be used to compare solution qualities of parallel and sequential algorithms, between best candidates of two algorithms and best-known solutions for the problem on hand. RPC formula for minimization problems is given below:

$$RPC = \frac{X_{New} - X_{BestKnown}}{X_{BestKnown}} \times 100$$

$X_{BestKnown}$ is the best solution found for a problem instance in literature, while X_{New} is the solution that needs to be evaluated. Formula above will yield the difference of newly found solution compared to best known solution to the problem on hand by percentage. A negative value indicates that the compared algorithm found a better solution than the best-known solution.

This will ultimately allow to measure how an algorithm performs compared to another algorithm. In addition, it will allow to measure how a change in a parameter affects the efficiency of parallel evolutionary algorithm, because finding the correct parameters is a key point for successfully discovering a wide area of search space and avoiding premature convergence.

CHAPTER 3

LITERATURE REVIEW

3.1. Literature Review on Quadratic Assignment Problem

Since its first proposal by Koopman and Beckmann in 1957 (Koopmans & Beckmann, 1957), to mathematically model economical activities, Quadratic Assignment Problem (QAP) gained a lot of attention on scientific world because it can be used in many applications.

Perhaps one of the most famous applications is “Steinberg Wiring Problem” described by Leon Steinberg to model placement of computer components in an electronic blackboard in order to minimize total amount of wiring needed (Steinberg, 1961). In his article he suggested three instances with a matrix size of 36 x 36. It took about 40 years of research to solve these famous problem instances. According to QAPLIB (Burkard, Karisch, & Rendl, 1997) , “b” and “c” instances were solved and proved to be optimal by Nyström in 1999 (Nyström, 1999), while “a” instance is solved and proved to be optimal by Brixius and Anstreicher in 2001 (Brixius & Anstreicher, 2001).

In 1974, Richard Francis, Leon F McGinnis Jr and John A. White published a book, in which they analyzed and modeled a decision framework for assigning facilities like police stations, schools and medical facilities on locations, according to the latest survey they utilized QAP for their framework (Francis, Jr., & White, 1974).

In 1972, Jakob Krarup created a dataset called kra30a to represent layout of newly built Klinikum Regensburg in Germany using QAP. His objective was to find a layout of hospital facilities in such a way that communication cost x distance between facilities were minimized (Krarup & Pruzan, 1978). Optimum solution for the problem is found with Branch and bound algorithm by Hahn, Hightower, Johnson, Spielberg, Roucairol in 2001; twenty-nine years later after it was proposed (Hahn, Hightower, Johnson, Guignard-Spielberg, & Roucairol, 2001).

In 1977, Dickey and Hopkins used QAP to model layout of buildings within a university campus (Dickey. & Hopkins, 1972).

In 1977, Burkard and J. Offermann tried to find the best possible typing machine keyboard using typing-time of an average steno typist and frequency of pairs of letters in different languages data (Burkard & Offermann, 1977).

In the same year, Alwalid N. Elshafei used QAP to find a layout for 19 hospital departments in such a way that minimizes the total distance travelled by patients (Elshafei, 1977) . In 1976, Hubert and Schultz used QAP for data Analysis (Hubert & Schultz, 1976).

In 1988, Laporte and Mercure used QAP to formulate a solution which minimizes the distance between the center of mass of the blades and the geometric center of the cylinder in hydraulic turbine runners by locating the turbine blades (Laporte & Mercure., 1988). In 1994, Phillips and Rosen used QAP to solve molecular conformation problem in their research (Phillips & Rosen, 1994).

In 1997, George and Pothen used QAP to formulate 1-sum and 2-sum problems to analyze spectral envelope reduction (George & Pothen, 1997). In 2015, Azab used QAP to model process planning mathematically in his work (Azab, 2015).

In 2017, Bougleux, Brun, Carletti, Foggia, Gaüzère, and Vento used QAP to compute graph edit distance measure (Bougleux, ve diğerleri, 2017).

According to a recent survey on subject (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007), QAP has also derivative problems like Linear Assignment Problem which is P class and can be easily solved by Hungarian Method. 3-Index Assignment problem, Quadratic Bottleneck Assignment Problem, Quadratic Three-dimensional Assignment Problem, Quadratic Semi-Assignment Problem, Multi objective Quadratic Assignment Problem all are special cases of the original problem.

There are three surveys on QAP in the literature, published in 1994, 1998 and 2007. Oldest survey belongs to Pardalos, Rendl and Wolkowicz, which was published in 1994 (Pardalos, Rendl, & Wolkowicz, 1994). It starts with formulizations of QAP and continues with derivatives of the problem, computational complexity of QAP, lower and upper bounds, introduces a local search algorithm, talks about exact solution methods and sub-optimal algorithms for QAP and introduces a library called QAPLIB. The Second survey belongs to Çela in the form of a book, written in 1998 (Çela, 1998). He starts with problem definition and continues with formulations of QAP, its

computational complexity, exact algorithms and lower bounds, heuristics on QAP and finishes with derivatives of QAP.

The latest survey to Loiola, De Abreu, Boaventura-Netto, Hahn, Querido and is published in 2007 (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007). It starts with brief history of QAP, continues with its formulations found in literature, QAP-related problems, lower bounds found in literature, exact, heuristic and metaheuristic algorithms for solving QAP and finishes with main research trends and tendencies.

Many data sets were proposed by different scientists all over the world since proposal of QAP. There are mainly 15 data sets used by researchers, all of which can be found in QAPLIB, a library dedicated to Quadratic Assignment Problem. The library contains problem instances, their optimal and best-known solutions, best solution methods, brief histories about data sets, when they are proposed and for what purpose. It was originally created by Stefan Karisch in 1997 with the help of Burkhard and Rendl. He maintained the relevant web site until 2002. From 2002 to 2012 website was maintained by Erande Cela. After 2012 website is maintained by Peter Hahn and Miguel Anjos. Last bulk update to the website was done in 2011 (Burkard, Karisch, & Rendl, 1997).

Quadratic Assignment problem is proven to be NP-Hard by Sahni and Gonzales back in 1976 in their article “P-Complete Approximation problems” (Sahni & Gonzales, 1976). This fact has led scientists all over the world to consider two possible approaches when dealing with quadratic assignment problem. Some scientists followed the exact methodologies and they were able to come up with optimum solutions for small or specific instances of the problem. Others decided to relax the finding optimum requirement and settled with sub-optimal results and utilized heuristic and meta-heuristic methodologies to solve the larger and more general problem instances to some degree. In the literature, between 1957 and 1990’s generally first approach was more common. While after 1990’s with metaheuristics became more popular, second approach became more common. There are different exact and heuristic/metaheuristic methodologies to solve quadratic assignment problem in the literature.

3.1.1. Exact Algorithms in the Literature

Today, exact algorithms are considered to be effective for instances smaller than size of 30 and some bigger instances for only specific instances. With the developments in computer hardware, and parallelization of algorithms, this threshold will increase more and more.

The most popular of the exact methodologies when solving QAP is branch and bound algorithm. One of earliest applications of branch and bound method to QAP was in 1966 by Gavet and Plyter (Gavett & Plyter, 1966).

In 1979, Bazaara and Elshafei created a branch and bound algorithm in their work which could solve instances up to size of 20 (Bazaraa & Elshafei, 1979). In 1980, Edwards proposed a binary branch and bound algorithm for the solution of Koopmans-Beckmann QAP (Edwards, 1980).

In the same year, Burkhard and Derigs proposed a branch and bound based solution to the Nugent 5, 6, 7, 8, 12 and 15 instances and become the first one to solve Nugent15 instance (Burkard & Derigs, 1980). In 1983, Bazaraa and Kirca proposed a branch and bound algorithm that can solve Nugent12a and Elshafei19 instances (Bazaraa & Kirca, 1983).

In 1987, Roucairol proposed a parallel branch and bound algorithm which can work on a multiprocessor computer or a cluster using a common shared memory. She tested her algorithm on a Cray X-MP machine and coded her algorithm in FORTRAN. She tested her algorithm on Nugent instances with size 5, 6, 7 and 8 with 1, 2 and 3 processors and calculated speed-up for each of them. She found out that the speed-up for small instances like 5 and 6 were negligible, while for bigger instances like 8 or 10, speed-up values were almost equal to the number of processors added (Roucairol, 1987).

In 1989, Pardalos and Crouse suggested another parallelized branch and bound algorithm which was essentially an enhanced version of Roucairol's work. They also worked on Nugent data set with instances of size 5 to 15, 20 and 30. They noted that, their algorithm only worked on symmetric instances and not in asymmetric ones. They were able to find the exact solutions for instance sizes up to 15. For 20 and 30 size instances, the exact solutions were not found. They were able to obtain speed-up ratios comparable to Roucairol's work (Pardalos & Crouse, 1989).

In 1993, Laursen adopted parallelized branch and bound into several problems, including QAP (Laursen, 1993). In 1997, Brüngger, Marzetta, Clausen and Perregaard proposed a parallelized branch and bound algorithm based on ZRAM parallel search library (Brungger, Marzetta, Clausen, & Perregaard, 1997). They were able to gain linear speedups and were able to solve 10 previously unsolved instances from QAPLIB. Instances are had16, had18, had20, tai17a, tai20a, rou20, nug21, nug22, esc32e, esc32f.

In 2001, Anstreicher, Brixius, Goux and Linderoth proposed a branch and bound algorithm which solves previously unsolved nug27, nug28, nug30, kra30b, kra32 and tho30 instances (Anstreicher, Brixius, Goux, & Linderoth, 2002).

One of the most famous set of QAP instances are called Nugent instances. According to a recent survey on QAP, all of these instances are solved to optimality by branch and bound methods except instance size 8 (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007). Below is a detailed table showing which instance size of Nugent data set is solved to optimality, by whom and in what year and using which methodology. Table 3.1 clearly shows most commonly used exact methodology to obtain the optimal solutions is branch and bound strategy.

Table 3.1. Nugent Instances and who solved them using which methodology. (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007).

Instance Size	Who Solved? -When?	Method
8	Burkard - 1975	Complete Enumeration
12	Burkard & Stratman - 1978	Branch & Bound
15	Burkard & Derigs - 1980	Branch & Bound
20	Colorni et al. - 1996	Branch & Bound
21 - 22 - 24	Brüngger et al. - 1997	Branch & Bound
25	Hahn – 2000	Branch & Bound
27 - 28 - 30	Anstreicher et al. - 2002	Branch & Bound

Another exact algorithm that is used when solving QAP instances is called Dynamic Programming. It is only used in a special case called Tree QAP where non-zero entries in flow matrix forms a tree. It is an extension to the branch and bound methodology

which was introduced by Christofides and Benevent in 1989 (Christofides & Benevent, 1989).

Another type of exact strategy is called cutting plane method which was introduced by Bazarara and Sherali in 1982 (Bazarara & Sherali, 1982). According to the recent survey, this methodology is only proper for small size instances of QAP due to its slow convergence. Bazarara and Sherali tested this method on Nugent 5, 6, 7, 8, 12, 15, 20 and 30 instances, Steinberg 36 instance, Elshafei 19 instance and Krarup 32 instance. An enhanced version of this algorithm is proposed by Padberg and Rinaldi named branch and cut technique seems to accelerate algorithm's convergence, although work was focused on symmetric traveling salesman problem and not quadratic assignment problem in general (Padberg & Rinaldi, 1991).

3.1.2. Metaheuristic Algorithms in Literature

Due to the disadvantages of the exact algorithms, many large - sized as well as some smaller - sized problems cannot be solved optimally. This mandates some relaxation from finding the optimum solution and settling with a "good enough" solution in a "reasonable time" frame for many instances of QAP. Many researchers focused on metaheuristics in the past, because they are easy to adapt to different problems and generally find good solutions. Since QAP is considered as a benchmark problem, it is perfect to test how successful a certain metaheuristic is compared to the other algorithms. Many metaheuristics have been proposed in literature for solving Nugent and other QAP instances on QAPLIB. Simulated Annealing, Genetic Algorithms, Ant Colony Optimization, Neural Networks, Tabu Search, Greedy Randomized Adaptive Search Procedure, Variable Neighborhood Search, Iterative Local Search, Breakout Local Search, Particle Swarm Optimization and combinations or derivatives of these approaches can be found commonly.

One of the earliest examples is Burkard and Rendl's simulated annealing solution in 1984. Authors state that they could reach solutions which differs from the optimum values by 1% or 2 % and they concluded that this could be an acceptable solution with smaller hardware demand with a shorter time frame (Burkard & Rendl, 1984).

In 1987, Davis compared Genetic Algorithms and Simulated Annealing solutions to QAP in his book as an example of combinatorial problems (Davis, 1987).

In 1990, Skorin-Kapov published another article in which she solved Nugent 15, 20 and 30 instances, Krarup 30, Steinberg 36 instances, as well as her own instances with sizes 42, 49, 56, 64, 72, 81 and 90 using Tabu Search method (Skorin-Kapov, 1990).

In 1991, Taillard proposed another Tabu Search algorithm which is implemented as a parallel algorithm and solved Nugent 5-30 instances, Elshafei 19, Krarup 30, Steinberg 36, Skorin-Kapov 42, 49, 56, 64 Instances and another data set that he created using parallel Tabu Search. He claimed with this method some problems can be solved up to size 64 with sub-optimal but good results (Taillard, 1991).

In 1993, Chakrapani and Skorin- Kapov created first massively parallel taboo search implementation on supercomputer Connection Machine 2. Algorithm utilized n^2 cores where n is the size of problem. Their work computed solutions for problems with sizes up to 90, as the best known or close to the best known, while for sizes bigger than or equal to 100, best known solutions are improved. (Chakrapani & Skorin-Kapov, 1993)

In 1993, Fleurent and Ferland published a parallel hybrid metaheuristic which combines local search and Tabu Search with Genetic Algorithms. In their paper, they compared 5 different algorithms; a local search procedure with pairwise exchange (LS), a Tabu search algorithm that was run $4n$ times and $4000n$ times respectively (Tab $4n$ and Tab $4000n$), same local search and Tabu algorithms with different starting points 1000 times (RepLS and RepTab $4n$) and a Genetic Hybrid Algorithm with LS and Tabu Search (GHA) on Skorin-Kapov 72-100 instances. Their results indicated that GHA performs better when compared to single Local Search or Single Tabu Search algorithm in almost all instances. They reported that their algorithm has improved best known scores for Taillard 40, 50, 60 and 80 instances and Skorin-Kapov 81, 100a, 100b, 100c, 100d and 100e instances (Fleurent & Ferland, 1993).

In 1994, Li, Pardalos, Resende used a metaheuristic named as greedy randomized adaptive search procedure (GRASP). GRASP is an iterative search procedure which has two phases. In the first phase, also named as construction phase, algorithm iteratively constructs a candidate solution by adding an element to the permutation or assigning a facility to a location. Choice of next element is determined by a greedy fitness function. This function did not necessarily select the best possible choice but one of best from a number of solutions. This allowed algorithm to be stochastic in nature. The solution produced in first phase is not guaranteed to be locally optimum.

Therefore, the algorithm goes over a second phase called local search. In this phase, a two-opt local search procedure is applied to the final solution. They tested their algorithm on almost all instances of QAPLIB; Burkard, Christofides, Elshafei, Eschermann, Krarup, Nugent, Roucairol, Scriabin, Steinberg and Li & Pardalos. The algorithm was one of the most-successful metaheuristics in literature; it was able to find almost all best-known solutions as well as improving some of those solutions (Li, Pardalos, & Resende, 1994).

Between 1994 and 1999, Maniezzo and Colomi adapted several versions of ant colony algorithm for solving QAP (Maniezzo & Colomi, 1994). Their series of work first started in 1994 where they adopted ACO algorithm developed to solve traveling salesman problem to QAP for the first time in literature. This algorithm employed a number of intelligent computing agents called ants which construct a solution incrementally by assigning facilities to locations. To avoid assigning same facilities to different locations, they added a Tabu list which included forbidden moves in each step. They tested their algorithm on Nugent 15-30, Elshafei 19, Krarup 30 instances. They used optimal solutions as a comparison base for instances smaller than 20, while for two larger instances (Nugent30, Krarup30) they used the best-known solutions computed by Burkard et al. (Burkard, Karisch, & Rendl, 1997). They compared their algorithm with GRASP, another heuristic developed by Pardalos et. al in 1994 (Li, Pardalos, & Resende, 1994). Their results were comparable to GRASP. Although a single ant would quickly converge on a local maximum point, collaboration between several ants yielded a good set of solutions.

In 1994, Nissen devised a sequential Evolutionary Strategy (ES) algorithm to solve QAP. He used permutation representation and standard random population generation. After that, in each iteration, children are created by applying between 0 to 2 random swap mutations to the parents, since he determined standard mutation scheme is not applicable in this problem. He also did not use any recombination operators. Algorithm was able to adapt mutation probability based on success of children compared to their parents. If for a couple of generations, children are less suitable, then mutation amount is automatically increased to widen the search area. He then compared his solutions to Simulated Annealing of Heragu and Alfa (Heragu & Alfa, 1992) and Tabu Search Algorithms of Taillard (Taillard, 1991) and his own implementation of two-opt search on Nugent 20 and 30 instances, Skorin - Kapov 64, Elshafei 19 and Steinberg 36

instances. He found out ES algorithm performed better than two-opt, while it was on par with TS and SA (Nissen, 1994).

In 1995, Bousoño-Calzón and Manning applied a Hopfield Neural Network to solve QAP. According to the authors, this type of neural network is often applied to the optimization problems, because main purpose of this network is to maximize the throughput while minimizing the cost; therefore, it might outperform conventional methods. However, this only works if the instances in question are symmetrical. Authors used this problem to represent processors, their distances and communication weights and come to a solution implementing a neural network to minimize the sum of all communication costs within a multiprocessor environment (C.Bousoño-Calzón & Manning, 1995).

In 1995, Tate and Smith published a sequential genetic algorithm to solve quadratic assignment problem. They designed 10 runs containing different random seeds for each instance, to assess overall performance of algorithm. Their algorithm had 3 different setups:

- 25% of population replaced by children and 75% of parents undergo only mutation
- 50% of population replaced by children and 50% of parents undergo only mutation
- 75% of population replaced by children and 25% parents undergo only mutation.

They limited population size to 100 and determined optimum found or 2000 generations reached as termination condition. They worked with a total of 11 Nugent instances. They determined setup with 25% of population replaced by children and 75% of remaining population undergo mutation as the best by a small margin. They found out higher mutation rates are more successful when solving QAP. They also reported that their algorithm found comparable results to previously known metaheuristics (Tate & Smith, 1995).

In 1996, Bachelet et al. tested different metaheuristics to compare their performance on QAPLIB instances. All the compared algorithms were parallelized algorithms. For the base case, they implemented a sequential hill climbing search algorithm with replace best strategy. For the Parallel Tabu Search, they implemented an algorithm which runs multiple independent tabu search algorithms with no communication.

Algorithms are initialized with different solutions and different parameters in the beginning. For genetic algorithm, they used a cellular model where the main population is divided into sub-regions called cells and the interactions between two individuals are limited to their neighbors. Mutations are applied to the individual cells and each individual is evolved on separate processors. They tried this setting in both SIMD and MIMD architectures. They claimed that, Cellular GA is better at exploration and Tabu Search is better at exploitation. Therefore, they also proposed two types of parallelized hybrid algorithms. In Parallel Synchronous Hybrid (PSH) Algorithm, they utilized Tabu search as a mutation operator for Cellular GA. In Parallel Asynchronous Hybrid (PAH) Algorithm, they proposed a system where algorithms exchange solutions on certain time intervals. They compared their results with ES from Nissen (Nissen, 1994), SA from Heragu and Alfa (Heragu & Alfa, 1992) and Tabu Search from Taillard (Taillard, 1991) on Nugent 30, Steinberg 36 and Skorin - Kapov 64 instances. They concluded that, while each algorithm has its own strengths, overall GA - Tabu Search hybrids perform better than others on average (Bachelet, Preux, & Talbi, 1996).

In 1997, Cung et al. proposed a Scatter Search procedure to solve QAP. This procedure is similar to an evolutionary algorithm, because both works on a randomly generated population. Scatter Search algorithm then combines several elite solutions (more than two) from main population and tries to find a good trial point. Once they obtained results from scatter search algorithm, they improved results further with a Tabu search algorithm which are integrated into the main population of scatter search algorithm. They executed their algorithm using Elshafei, Taillard, Krarup, Skorin – Kapov and Thonemann instances. Authors reported that, their algorithm was able to find best-known or optimal values for most of problem instances and was able to improve best - known solutions for tho150 and tai256c instances (Cung, Mautor, Michelon, & Tavares, 1997) .

In 1999, Stützle applied Iterative Local Search (ILS), yet another metaheuristic to QAP. This algorithm first generates an initial solution and applies a local search procedure on that solution, using two-opt algorithm. Then, in the next step, the algorithm perturbrates the obtained solution, applies the same local search again and decides whether it meets the termination criterion. Author also pointed out that, it is possible to make ILS a population based procedure by maintaining set of solutions instead of a

single one. He concluded that, while population based ILS algorithm gives better results in overall, single solution ILS implementations are open to improvements and overall results of the algorithm are promising (Stützle T. , 1999)

In 1999, Gambardella, Taillard and Dorigo created another Ant Colony algorithm which uses an extra local search procedure. In a standard ant colony approach, pheromone trails are manipulated by agents during or after the construction of new solutions locally as well as globally. However, in this new approach, pheromone trails are only updated globally. This drives the algorithm towards a quick converge to the solution. Downside of this approach is, sometimes premature convergence may happen and solution may be sub-optimal. Authors therefore developed a diversification mechanism which erases pheromone trails periodically. Additionally, in previous applications, pheromone trails were used to construct new solutions while in their implementation, those trails are used to improve existing solutions (Gambardella, Taillard, & Dorigo, 1999).

In 2000, Ahuja, Orlin and Tiwari suggested a genetic algorithm based on greedy principles like initializing starting population with a metaheuristic algorithm namely GRASP technique suggested in 1994 by Li et. al. (Li, Pardalos, & Resende, 1994). For the parent selection they did not employ a fitness proportionate strategy, rather selection was based on uniform random probability for each solution. They created three new crossover schemas and found the most effective to be shift-path crossover in which, identical portions of the chromosome in parents are determined and other parts of chromosome of one parent is shifted until the most similar fitness value to the other parent can be obtained. After that, selection of the survivor depends on the created children. Since this technique only allows selection of single parent, if child is fitter than both parents, the one that is the most similar to child is eliminated and if there is a fitter parent compared to child, that parent is selected. For mutation part, they employed an algorithm which creates an $n \times n$ matrix, in which value at the position ij defines the number of individuals in the current population in which site i contains the facility j . For the local search part of the algorithm, they employed a 2-exchange neighborhood algorithm to improve the already found results. They also employed a strategy called tournament in which they gathered half of the final population from two different runs, merged into a single population and re-run the algorithm. They tested their algorithm on 132 instances from QAPLIB including Burkard, Elshafei,

Christofides and Eschermann and determined their algorithm was able to find 103 of the 132 best known solutions and for the remaining, it was able to approach 1% to the best-known solution (Ahuja, Orlin, & Tiwari, 2000).

In 2001, Talbi, Roux, Fonlupt, Robillard proposed an ant colony solution which is strengthened by Tabu search algorithm. In their system, each ant iteratively creates permutations of QAP, better the solution finds, more pheromone trail it produces. Solutions produced by ants are then used in a Tabu search algorithm to improve global solution quality. They compared their algorithm with another ant colony implementation from previous years, HAS-QAP created by Gambardella et al. in 1997 (Gambardella, Taillard, & Dorigo, 1999) using Nugent, Skorin-Kapov and Taillard instances. Their algorithm named as, ANTabu outperformed HAS-QAP on all test cases (Talbi, Roux, Fonlupt, & D.Robillard, 2001).

In 2003, Misevicius (A.Misevicius, 2003) proposed a simulated annealing algorithm which he hybridized with Tabu Search algorithm to improve the results. He started with random permutations, for the neighbourhood discovery scheme, he applied a 2-way perturbation which swaps two elements until all possible swaps are applied. Neighbours are searched based on a fixed order. In total, $n(n - 1)/2$ trials are done to search all the neighbourhoods. When a better solution is found, his algorithm replaced it with the current solution and implemented the same procedure again. Before starting the annealing procedure, his algorithm considers a few random moves and gets the average difference, relying on the idea that the average fitness value directly affects initial and final temperatures. Another improvement he made is applying an adaptive, dynamical cooling schedule to his algorithm. After each annealing step, his algorithm improves best found result using CRAFT algorithm (Armour & Buffa, 1963). After certain number of iterations, his algorithm terminates. He then utilized Tabu search on the best result. His implementation is based on Taillard's work in 1991 (Taillard, 1991). He compared his algorithm with previous simulated annealing implementations in the literature: Connolly's algorithm (C-SA-QAP) (Connolly, 1990) and Bölte and Thonemann's algorithm (TB2) (Bölte & Thonemann, 1996). He started all algorithms with the same set of parameters and the same initial permutations to measure his algorithm. He concluded that his algorithm surpassed both Connolly's algorithm and Thonemann's algorithm based on the QAPLIB instances: Krarup, Nugent, Skorin-Kapov, Steinberg, Taillard, Thonemann

and Wilhelm. He also pointed out that, his algorithm improved the best-known solution for Thonemann 150 instance.

In 2004, Misevicius suggested another algorithm, this time hybridization of Genetic algorithm and Tabu search (A.Misevicius, 2004). He used Tabu search as a mutation operator to improve solutions. He also adapted a mechanism to detect premature convergence at the end of each generation. For the parent selection, he used rank selection methodology. For the cross-over operator, he used an improved version of uniform like crossover (ULX). Idea is that, identical genes on both parents are first copied into child and remaining genes are replaced from both parents' uniformly and randomly. His improvement creates more than one child and selects the best among them. He tested his algorithm on QAPLIB instances; Taillard, Eschermann, Krarup and Steinberg. He compared his algorithm with Taillard's Tabu search in 1991 (Taillard, 1991). He reported that, his algorithm obtained best-known results except for Taillard 60, 80 and 100 instances.

In 2005, Misevicius proposed yet another algorithm, this time an implementation of Tabu search. He pointed out in his paper, like many other metaheuristics, Tabu search also suffers from a condition known as stagnation in which algorithms starts uncovering very large domain in solution space and converging from a local area very fast and then as the search progress, it takes extremely long time to improve due to lack of diversification. He proposes using Tabu search as a tool for intensification and another mutation strategy to diversify the solutions obtained. He used Taillard's Tabu algorithm (Taillard, 1991) and modified it to encompass an additional mutation procedure. Author found out this version of Tabu search yields slightly better results than the original version. He tried four different versions of mutation. The first was swap mutation in which two genes of genome (permutation) are randomly swapped. In the second model, entire genome is divided into two equal sized parts from middle point and random swaps are made between two random genes from both sides. In the third model, shift mutation, where all elements are shifted by a given number. In the fourth model, two randomly selected neighbours are interchanged. He performed experiments solely on Taillard data set, both with symmetric 'a' instances and asymmetric 'b' instances. He found out that, for smaller instances like tai20a, tai25a and tai30a, shift mutation performed better, while for larger instances like tai50a,

tai60a and tai80a, neighbour exchange method performed better, while all of the results were competitive (A.Misevicius, 2005).

In 2006, Tseng and Liang proposed a hybrid metaheuristic which combines genetic algorithms and ant colony optimization in an algorithm named ANGEL. Their algorithm consists of two stages. In the first stage, ant colony algorithm runs and creates an initial population for genetic algorithm. In the next stage, genetic algorithm kicks in and improves the sub-optimal solution pool. They used a variant of List Order Crossover and added a secondary crossover applied to best known solution with others in population named as eugenic crossover. In addition to genetic algorithm, in final step, a two-opt local search procedure is used. After genetic algorithm terminates, ant colony algorithm once again starts and tries to diversify population again for next stage of genetic algorithm. This cycle continues until best-known solutions are obtained. Authors reported that they tested their algorithm on 100 QAPLIB instances and they were able to find optimum or best-known solutions for 90% of the instances (Tseng & Liang, 2006).

In same year, Demirel and Toksari proposed another hybrid algorithm which combines ant colony algorithm with simulated annealing as a local search procedure, which they named as AntSimulated (Demirel & Toksari, 2006). Their algorithm starts as a normal ant colony procedure, a number of solutions are generated by each of the ants and all pheromone trails are set to same value. Then, each of these solutions are improved in two stages: pheromone trail based modification and a local search based on simulated annealing. In the first step, a number of swaps based on pairwise exchange inside permutation is applied to each of the solutions based on the pheromone traits. In the latter stage, three best found solutions are transferred to secondary simulated annealing procedure and algorithm runs for 200 iterations with an initial temperature of 100 and final temperature of 10. Neighbourhood is defined with either swap mechanism or objective swap mechanism which considers all possible swaps for elements inside permutation and selects the best. In the next stage, improved solutions are returned to the ant colony algorithm, evaporation and update of pheromone trails are done after that algorithm checks whether stopping criterion is satisfied. If the criterion is not satisfied, ant colony algorithm continues on next iteration. In the end, they compared their algorithm with Gambardella et al.'s HAS-QAP algorithm (Gambardella, Taillard,

& Dorigo, 1999) on QAPLIB instances and reported AntSimulated performs better on most instances.

In 2006, Stützle proposed yet another Iterated local search (ILS) algorithm for QAP. He pointed out that in order to apply ILS to QAP, one must consider four fundamental functions:

- GenerateSolution: Creates a random solution
- Perturbation Function: Applied to a solution to create another solution to discover other parts of search space by randomly exchanging parts of permutation
- A Local Search Function: to improve existing solutions. Two opt search is widely used.
- AcceptanceCriterion: Whether to accept current result or not.

Although simple, Stützle points out that ILS is an effective metaheuristic in TSP-like problems. He run several experiments to determine its effectiveness for QAP. He found out that ILS algorithm was prone to stuck in a small area of search space and cannot explore different regions despite high perturbation rate. He proposed two types of solutions. One of them includes a soft restart procedure in which, whenever algorithm decides a stagnation happened, model restarts itself from another point in search space. Another solution he proposed was to make ILS population based instead of single solution based, which might increase diversity and improve algorithm. Author compared his algorithm with previous ILS based implementations, like Taillard's Robust Tabu search (Taillard, 1991). He reported that, current ILS with soft-restart performs better in all instances. He also reported that, population based ILS algorithm performs better (Stützle, 2006).

In 2009, James et al. suggested a parallelized Tabu Search algorithm to solve QAP. The cooperative parallel tabu search algorithm (CPTS) as the authors name it, is implemented with the idea of running multiple TS algorithms on different processors, while communication among different algorithm instances share a global long-term memory in which they store their solutions. Half of the solutions are the best-found ones so far in any stage of the algorithm. This guarantees the half of the TS algorithms start with copy of the best-found solutions, therefore it drives the common search

process to the more quality areas of the search space. Each of the algorithms are assigned with different parameters to ensure diversification to some degree. There are reference sets in memory associated with each of the TS processes. In the start of the algorithm, a single instance of TS runs on all processors to update reference sets associated with them. This provides seeding the start of algorithm with relatively good solutions and some diversification. In the next step, each processor runs its own TS algorithm based on the results from their reference sets. When each of the algorithm terminates, a global algorithm updates the entry in reference set and checks whether the solution updated is the best-known solution. If it is, this solution is distributed to the other algorithms in other processors via shared reference set. To maintain diversity, the global algorithm also checks if the solution obtained is changed each time. If it did not, the algorithm starts main diversity operator to reseed the local algorithm on that processor. The termination of the global algorithm is determined by number of iterations. Authors tested this algorithm with Taillard and Skorin-Kapov instances from QAPLIB. Also, they compared their algorithm with 10 other implementations from literature including Tseng and Liang's GA +AC hybrid from 2006 (Tseng & Liang, 2006) and Misevicius' tabu search algorithm from 2005 (A.Misevicius, 2005) and Misevicius' genetic algorithm - tabu search hybrid from 2004 (A.Misevicius, 2004). Authors indicated that, each of the algorithm performances varies depending on the data set used. If an algorithm runs well on Taillard's set, it does not guarantee to run well on Skorin-Kapov instances. However, they pointed out that, their algorithm matches or outperforms other algorithms in terms of solution quality, except in some special cases. For instance, Misevicius' tabu algorithm performed slightly better on tai100b instance (James, Rego, & Glover, 2009).

In 2013, Benlic and Hao proposed a breakout local search (BLS) algorithm to solve QAP (Benlic & Hao, 2013). Their algorithm works similarly to an Iterated local search algorithm. Algorithm starts with an initial solution and iterates between two phases; local search and perturbation. In local search phase, algorithm performs a steepest descent local search algorithm on solution at hand and obtains an improved solution. If the algorithm cannot improve the solution, it assumes it is a local optimum and proceeds with perturbation process to explore a different region in search space. If exploration of search space is weak, algorithm might be stuck on a stagnation point where it ends up cycling between couples of local optimum points and cannot improve.

But, if the diversification procedure is strong, algorithm might get a random restart and waste considerable amount of time. Authors found solution to this problem by introducing an adaptive perturbation technique. Idea with adaptive perturbation is that, algorithm performs a small change on permutation and tests whether it is strong enough to jump to another local optimum area. If the jump is not sufficient, amount of change to the original permutation is increased. They pointed out adaptive perturbation scheme is the most important factor which gives BLS advantage over ILS algorithm. BLS also borrows tabu list notion from tabu search to avoid circling between the same points, although BLS does not consider exploration in intensification phase, while SA and Tabu search does. Authors also experimented on different QAPLIB instances to see its effectiveness. They compared their algorithm to CPTS by James et al. (James, Rego, & Glover, 2009), Misevicius' tabu search algorithm (A.Misevicius, 2005), and Misevicius' genetic algorithm hybrid from 2004 (A.Misevicius, 2004) and a few others. BLS was able to solve every real-life like instance from QAPLIB except Taillard150b to their best-known values. For unstructured instances, it was able to find 7 of 9 best-known solutions and it is comparable to those given algorithms in terms of solution quality.

In 2013, Tosun, Dokeroglu and Cosar proposed an Island Model genetic algorithm for solving QAP (Tosun, Dokeroglu, & Cosar, 2013). To best of my knowledge, this is the only example in literature directly related to this thesis topic. QAP-IPGA algorithm, as they name it, uses different subpopulations exploring different regions in search space, thus dramatically increases the speed algorithm converges to a solution. Island model also facilitates a migration model between subpopulations to maintain diversity. In this algorithm, every subpopulation is assigned a slave processor which does the genetic operations as well as assigned with the task of introducing newly created individuals to the population. Since they do not replace any individual solution within population, this process is causing an increase within global population. However, authors limited this increase by a ratio to original population, once that amount is reached, slave processor stops generating. Algorithm terminates when a certain number of generations is reached. They tested this algorithm with 3 different crossover operators and 3 different selection methodologies. They used Order Crossover, Partially Mapped Crossover and Cycle Crossover, as well as, Truncation Selection, Tournament Selection and Roulette Wheel Selection as selection strategies. Out of 9

different configurations, they found out Tournament Selection as the best selection strategy and Partially Mapped Crossover (PMX) as the best crossover methodology. This is the same configuration which is used in this thesis. They also implemented a parallel exhaustive search algorithm which is able to solve instances optimally upto size of 16 for comparison purposes. They tested their algorithm on QAPLIB instances using a computer cluster with 46 nodes with a global population count up to 30,000 individuals. They found out, there is a sharp increase in solution quality up until 8 islands. This is also consistent with my experiments. They reported QAP-IPGA was able to solve instances up to 30 to the best-known results and for instance sizes between 30 and 150 their algorithm was able to produce results with an average 1% gap. They concluded that while parallelization has a significant effect on solution quality for a genetic algorithm, they cannot increase the size of optimally solvable instances in QAP since this problem is exponential.

In 2015, Benlic and Hao proposed yet another genetic algorithm using breakout local search as a tool for local search operation (Benlic & Hao, 2015). Their algorithm starts with randomly generating an initial population and improving upon that population using breakout local search. After this step, genetic algorithm starts working on provided local optimum solutions. They utilized tournament selection as parent selection strategy, for the crossover operator, they preferred Uniform Crossover (UX) and used breakout local search algorithm as a way to optimize cross over operation's output. This algorithm is the one they developed for their article back in 2013. They also developed a replacement strategy to replace existing parents with newly created children. Replacement strategy decides based on the test whether there is an identical solution to the newly created children and whether there is an individual with worse fitness value than newly created children within population. If those two conditions hold, replacement with worst-fitness individual happens. Though they acknowledged this greedy strategy may lead to premature convergence. To solve this problem, at least in theory, they developed an adaptive mutation operator which tests for how many iterations the best-found solution did not change. If a stagnation is detected, mutation operator increases the amount of mutation from the minimum value to higher values until there is a change in the best-known solution. Once the change is achieved, mutation amount returns the initial minimum. They set the termination criteria to two hours of execution time and tested their work on QAPLIB instances. They compared

their algorithm to their previous BLS algorithm implementation in 2013, as well as, Misevicius' genetic algorithm implementation in 2004 (A.Misevicius, 2004) and parallel tabu search algorithm of James et al. in 2009 (James, Rego, & Glover, 2009). They evaluated the proposed algorithm on the set of 135 instances from the QAPLIB. They reported that proposed algorithm outperforms its local search component (BLS) and other compared algorithms and was able to obtain best known results for 133 instances.

In 2015, Tosun introduced a parallel island model genetic algorithm - tabu search hybrid (U.Tosun, 2015), in which an island GA algorithm is run to find a nearly optimal set of results, in order to create a seed population. In the next step, a diversification operator is applied to vary seed solutions. He reported that final seed solutions are often very close to the best-known solutions. In final step, results are enhanced by using a parallel tabu search algorithm. Introduced parallel GA is the same one he and his colleagues proposed back in 2013 (Tosun, Dokeroglu, & Cosar, 2013). It is an island parallel model using ring exchange with PMX crossover inside subpopulations, and using tournament selection. Diversification operator used in second step is an enhanced version of the same operator proposed by James et al. (James, Rego, & Glover, 2009). In the final step, robust tabu search algorithm is applied which is the most famous metaheuristic for QAP proposed by Taillard in 1991 (Taillard, 1991). He tested this hybrid algorithm on QAPLIB instances. He compared newly obtained results with his previous work, as well as Benlic and Hao's BLS algorithm (Benlic & Hao, 2013), Ant colony - GA hybrid of Tseng and Liang (Tseng & Liang, 2006), Tabu search of Misevicius (A.Misevicius, 2005) and GA -TS hybrid of Misevicius (A.Misevicius, 2004) on Skorin - Kapov and Taillard instances. He concluded that, newly created algorithm surpasses its competitors in terms of time and solution quality, while BLS of Benlic and Hao performs better in terms of time in Taillard instances.

In 2016, Hafiz and Abdennour proposed 5 different particle swarm optimization (PSO) variants for QAP (Hafiza & Abdennourb, 2016). Authors also pointed out PSO in its natural form is not suitable for combinatorial optimization problems like QAP. They made it clear that, in its natural form, PSO is continuous and depends on Euclidian distance between particles, therefore when applying the algorithm, one must make it discrete, change distance measure to make it applicable for QAP. Authors also point

out that PSO is applied for TSP which is a more specific form of QAP, where weight matrix does not exist. In the proposed algorithm, they used permutations to represent assignment of facilities to locations, where a full permutation is considered as a particle. In this approach, probability of facility selected for a particular location is represented by velocity elements. After each iteration, velocity is updated based on permutations quality. In this way, more promising placement probabilities are increased as the algorithm works through iterations. Like its other counterparts, PSO is also prone to premature convergence due to lack of diversity which might take place in latter stages of search. To solve this problem, authors considered restarting idea first proposed by James et al. in 2009 (James, Rego, & Glover, 2009). In the context of PSO, restart can be thought as reinitialization of particles' velocities. This allows to retain algorithm's experience and thus personal and global best-known solutions.

They proposed 5 different variants of the PSO algorithm. In first approach called Local best PSO (LPSO), sharing of information is limited to the immediate neighborhoods which allows particles to explore more within search space. The best knowns are obtained through neighbors and not globally. In the second approach, called Dynamic multi swarm PSO, swarm is divided into sub-swarms and information is only shared within subswarms for a certain number of iterations. After that, the swarms are regrouped and redistributed randomly. This mechanism allows sharing information globally, while allowing certain level of diversity. For other 3 approaches, a learning strategy is varied. In the original algorithm, swarm's best position and particle's own best position is used. Third approach, Unified Particle Swarm Optimization (UPSO), combines particle's own knowledge and swarm's collective knowledge over a unification factor. In the fourth approach, Comprehensive Learning PSO (CLPSO), each particle only learns from another single particle, if solution of particle does not improve, it automatically makes another random selection. In the fifth strategy, Fully Informed Particle Swarm (FIPS), a particle learns from its entire neighborhood instead of a single other particle. They tested their algorithm and its variants on QAPLIB instances. Their tests have shown that UPSO method achieves the best results on 35 different instances. They also compared their algorithm with other techniques including Taillard's Robust tabu search (Taillard, 1991) and Maniezzo and Colorni's ant colony algorithm (Maniezzo & Colorni, 1994). They concluded that, although their algorithm performed well, in terms of solution quality against others, Taillard's

algorithm was simply better. Their interpretation was solution modification techniques are more powerful than techniques based on solution construction in QAP.

In 2017, Sagban et al. proposed yet another ant colony – local search hybrid algorithm implementation for QAP and TSP (R.Sagban, Ku-Mahamud, & Abu Bakar, 2017). Their algorithm was based on Stützle and Hoos' ACO variant (MMAS) proposed for TSP and QAP (Stützle & Hoos, 2000). According to the authors, their enhancements to MMAS was based on utilizing two types of memory: a component based and a population based memory. Component based memory stores significant components found. Significance is tested with an algorithm according to a threshold by a heuristic function. This process is highly dependent on evaporation of pheromone trails. This component will later be used to guide the ant in their probabilistic construction. On the other hand, population based memory is related to entire population rather than fragments of the solution. This structure keeps a sample of population that consists of the improved solutions. This structure is updated after each local search iteration, if the solutions found are better than those already known. They used 3-opt local search with first improvement strategy for QAP variant of the algorithm. They compared their algorithm with other ant colony implementations in the literature. Their algorithm was able to solve 22 of the 40 instances from QAPLIB (Burkard, Christofides, Elshafei, Krarup, Taillard) to best known solutions, while outperforming MMAS algorithm and other variants they used.

In 2017, Aksan et al. proposed a parallelized break out search algorithm which utilizes Levenshtein Distance metric to check similarity between already discovered points and new starting locations. This process enhances exploration procedure by saving algorithm from re-searching already discovered areas and enables other areas to be explored. They also implemented parallel, independent runs of same algorithm and implemented a master - slave model using OpenMP library. Levenshtein Distance similarity metric is a measure that determines how close two strings are. Authors pointed out that it already has promising applications in spell correction, speech recognition, computational biology, DNA analysis, machine translation, information extraction and plagiarism detection. Therefore, they decided to use this measure to find a similarity ratio between two permutations. Their algorithm starts with a number of randomly generated permutations, with the restriction of being at least 30% different from each other. This similarity checking mechanism enables different runs of the

algorithm to start searching in different parts of search space. After algorithm starts, each thread starts searching the neighborhood of initially created solution using steepest descent algorithm. To minimize the cost, each portion of permutation is inverted until the best solution in the neighborhood is obtained. After that, it continues from this location to search new solution's neighborhood and if a stagnation is detected, algorithm makes a jump from this local optimum using perturbation phase which utilizes tabu search. The fitness calculations are utilized in each of slave threads, using master - slave model. Authors reported that, their algorithm was able to achieve almost linear speedup. They tested their algorithm on 59 QAPLIB instances. Their previous tests with similarity ratio value shown that 30% to be the most effective for Levenshtein Distance. They compared their algorithm with other state of the art algorithms from literature including Tabu Search by James et al. (James, Rego, & Glover, 2009), Ant colony - GA hybrid of Tseng and Liang (Tseng & Liang, 2006), Parallel hybrid algorithm of Tosun (U.Tosun, 2015), Memetic search algorithm of Benlic and Hao (Benlic & Hao, 2015). Authors reported that their algorithm was able to be one of the three best performing results according to their experiments with Taillard, Skorin - Kapov, Krarup, Steinberg and Eschermann instances (Aksan, Dokeroglu, & Cosar, 2017).

3.1.3. Current Situation in QAP

According to QAPLIB website, (Burkard, Karisch, & Rendl, 1997) which was updated in 2011 for the last time, 11 types of instances are solved to optimality out of 15 types of published there, so far. There are still Skorin - Kapov, Taillard, Thonemann and Wilhelm instances whose optimal solutions are unknown. All of the instances are bigger than or equal to size of 30.

Table 3.2 shows the instances solved to the best - known value using one of the metaheuristics techniques, as well as, how close solution is to the calculated lower bound.

Table 3.5. QAPLIB Instances with optimum values unknown & Best-known metaheuristics. (Burkard, Karisch, & Rendl, 1997)

Instance Name	Solution Method	Gap from lower bound (%)
Taillard – 30a	Robust – Tabu Search	6.12 %
Taillard – 35a	Robust – Tabu Search	8,48 %
Taillard – 35b	Robust – Tabu Search	14.52 %
Taillard – 40a	Robust – Tabu Search	9.43 %
Taillard – 40b	Robust – Tabu Search	11.43 %
Taillard – 50a	Iterated Tabu Search	11.09 %
Taillard – 50b	Robust – Tabu Search	13.79 %
Taillard – 60a	Tabu Search	22.59 %
Taillard – 60b	Robust – Tabu Search	10.82 %
Taillard – 80a	Iterated Tabu Search	22.20 %
Taillard – 80b	Robust – Tabu Search	12.28 %
Taillard – 100a	Iterated Tabu Search	24.86 %
Taillard – 100b	Robust – Tabu Search	10.78 %
Taillard – 150b	Genetic Algorithm	10.78 %
Taillard – 256c	Ant Colony Algorithm	2.03 %
Skorin – Kapov – 42	Robust – Tabu Search	5.56 %
Skorin – Kapov – 49	Robust – Tabu Search	5.91 %
Skorin – Kapov – 56	Robust – Tabu Search	5.37 %

Skorin – Kapov – 64	Robust – Tabu Search	5.70 %
Skorin – Kapov – 72	Robust – Tabu Search	5.38 %
Skorin – Kapov – 81	Genetic Algorithm	5.41 %
Skorin – Kapov – 90	Robust – Tabu Search	5.63 %
Skorin – Kapov – 100a	Genetic Algorithm	5.37 %
Skorin – Kapov – 100b	Genetic Algorithm	5.44 %
Skorin – Kapov – 100c	Genetic Algorithm	5.54 %
Skorin – Kapov – 100d	Genetic Algorithm	5.54 %
Skorin – Kapov – 100e	Genetic Algorithm	5.54 %
Skorin – Kapov – 100f	Genetic Algorithm	5.60 %
Thonemann – 40	Simulated Annealing	6.69 %
Thonemann – 150	Simulated Annealing	6.30 %
Wilhelm – 50	Simulated Annealing	3.52 %
Wilhelm – 100	Genetic Algorithm	3.15 %

3.2. Literature Review on Parallelization of GA

An early attempt for parallelizing genetic algorithm dates back to 1976 work of Bethke (Bethke, 1976). In his work, he used master – slave model to parallelize some portions of two kinds of genetic algorithm. One of them is a simple generational GA and other one is a steady state GA. After that, he measured performances of the algorithms. His conclusion was, generational GA was well suited to this model, while steady-state model made less efficient use of parallel processors but was still reasonably efficient in terms of multiprocessing.

In 1981, Grefenstette published a technical report on applying parallelization on genetic algorithms (Grefenstette, 1981). According to Cantu-Paz, (Cantú-Paz, 1995) he proposed four different types of parallelization models. Among those four models, three of them were master-slave architecture based, while the last model was multi-population based. First model included a sequential generation GA where fitness calculations were done in slave processors in each iteration. Second model was also a sequential GA, but there was no synchronization between slave processors; they received fitness calculations whenever they finished previous one. In third model, population was stored in a shared memory structure, where every slave processor takes one individual whenever it is idle and calculates fitness value for that individual independently from other processors. In fourth model, he proposed a multi population schema, where best individuals are shared among processors at the end of every iteration.

In 1985, Grosso proposed a multi – population model in his dissertation (Grosso, 1985). According to Cantu-Paz (Cantú-Paz, 1998) his objective was to simulate diploid individuals and a global population which consisted of five separate demes that exchange a fixed number of solutions. Rate of the migration was changed in different experiments to see the effect of migration rate. He discovered that, the rate of improvement in his five-deme model were faster than a single combined population.

In 1987, Tanese proposed a parallel genetic algorithm which utilizes hypercube topology as a migration topology for the first time (Tanese, 1987). According to Cantu-Paz (Cantú-Paz, 1998), she defined a neighborhood structure between demes as being in the same dimension with each other. Immigrants of the demes were selected from the best solutions and replaced the worst solutions of other demes. Migrations happened in fixed time intervals. According to Cantu-Paz (Cantú-Paz, 1995), she reported that parallel GA's results were comparable to serial version with the advantage of near-linear speedup.

In 1989, Tanese published another paper as a continuation for her work in 1987 (Tanese, 1989). According to Cantu-Paz (Cantú-Paz, 1998), she compared performances of a serial GA, a multipopulational parallel GA using a hypercube topology and multipopulational GA with no communication. All algorithms are run for 500 iterations. She concluded that, multipopulational parallel GA was able to find same solutions as the sequential version. However, final solution quality was worse.

Solution quality surpassed the sequential version with introduction of migration schema.

In 1990, Whitley and Starkweather created a parallelized version of their previously implemented GENITOR algorithm by using smaller distributed populations and a migration schema. Their implementation of the migration schema transfers the best individuals in each of the subpopulations to their immediate neighbors and replaces worst individuals in that population using a ring migration topology. They called this new algorithm GENITOR II and tested it on several problems, like optimizing a feedforward neural network and traveling salesman problem. They used three versions of the algorithm in comparisons to sequential version of the algorithm GENITOR, distributed version of the algorithm GENITOR II and another distributed version without a migration schema. They reported that, distributed algorithm is more robust and they were able to optimize larger size problems and they acknowledged distributed algorithm to be superior to the serial version in terms of solution quality (Whitley & Starkweather, 1990).

In 1991, Mühlenbein et al. applied a distributed genetic algorithm named as PGA to optimization of continuous functions. To avoid premature convergence and stagnation, they also implemented a hill – climbing algorithm for each subpopulation. For this application, they used a ring-like migration topology, which they call ladder topology. Each of the subpopulations evolve for a fixed period of time, utilizing a steady-state GA. Migration takes place after certain time intervals, and best individuals are migrated to the immediate neighbors of the subpopulations. The algorithm terminates after the algorithm reaches a certain fitness value. They benchmarked their algorithm on several different problems and measured a superlinear speedup in some cases. They also reported better solution quality and even found a global minimum point which was previously unknown for one of the functions (Mühlenbein, Schomisch, & Born, 1991).

In 1993, Whitley and Gordon compared 9 different parallel GA algorithms for continuous function optimization that were previously introduced in the literature. Among those algorithms, two of them were using master – slave model, four of them were coarse-grained and two of them were fine-grained algorithms. Other algorithms were two different implementations of Goldberg’s famous Simple Genetic Algorithm (Goldberg, 1988). This is the earliest work referencing coarse-grained algorithms as

an island model. They normalized the algorithms to use same number of fitness evaluations as a base for comparison. They used De Jong functions, Rastigin, Schwefel, Griewangk, Ugly 3, 4-bit deceptive functions and zero-one knapsack problem for benchmarking the algorithms. All the algorithms were started with a population size of 400. According to their test results, Simple GA derivatives got the worst results in terms of solution quality, while parallel implementations performed better. Most notably, Whitely and Starkweather's GENITOR II algorithm (Whitely & Starkweather, 1990) got the best results. They also noted that, elitist versions of the algorithms were better than non-elitist versions (Gordon & Whitely, 1993).

In 1994, Levine used an island model GA to solve Set Partitioning Problem in his dissertation. Set partitioning problem is a type of combinatorial optimization problem used by many airline companies for flight crew scheduling. He created a steady-state sequential GA and distributed it among N-Processors of an IBM SP-1 computer. He measured his results in terms of solution quality. He fixed the number of migration intervals to 1000 generations on each of the tests, while utilizing ring migration in which, a single best solution from each population replaced its immediate neighbor's worst solution. He used 1, 2, 4, 8, 16, 32, 64 and 128 islands on his experiments. He concluded in his experiments that; migration was preferable to no migration in island model. However, he was unable to find a significant difference among different migration intervals. He also concluded that, incrementing number of islands has contributed to solution quality for set partitioning problem (Levine, 1994).

In 1998, Ochi et al. applied an island model GA to vehicle routing problem. Vehicle routing problem consists of a number of vehicles with a fixed capacity that are to deliver items of different quantities to different customers. Knowing the distances between customers, objective of problem is to find total minimum distance traveled by the vehicles, given that only a single truck handles deliveries for a given customer and total quantity that must be carried by truck cannot exceed maximum capacity allowed. Their approach was combining a parallel island model genetic algorithm with scatter search. Their migration policy is unusual: when one of the islands starts to lose diversity, it issues a broadcast requesting best known solutions from all other islands. This allowed creation of somewhat different population and broadens the exploration of search space. They compared their results to Taillard's sequential algorithm and

they found out some improvements in terms speedup and solution quality. Their reported speedup gain was almost linear (Ochi, Vianna, Drummond, & Victor, 1998).

In 2002, Fan et al. applied an island model genetic algorithm to segment lateral ventricles from magnetic resonance brain images. By applying active model-based segmentation technique, researchers were able to convert the process into an optimization process. Then, an island model GA is run to solve the problem. In initial phase, rough descriptions of object surfaces are inputted as initial population to the genetic algorithm and algorithm is utilized to refine those images using an energy minimization procedure. Their algorithm included a somewhat unorthodox migration methodology, in which how many subpopulations will migrate solutions is determined by the algorithm randomly, while best known solutions of the islands are migrated. They noted that parallel genetic algorithm did a very good job as the results of experiments indicated that their approach surpassed existing sequential GA solutions with more effective discovery of search space and increased convergence speed gained by the parallel island model GA (Fan, Jiang, & Evans, 2002).

In 2003, Pereira and Lapa applied an island model GA to nuclear reactor core design optimization problem. This optimization targeted minimizing the average-peak factor in a 3-enrichment-zone reactor by adjusting several reactor cell parameters, such as dimensions, enrichment and materials. They used a fixed size 50 intra-island populations and utilized ring migration strategy where best – individuals are migrated at every 10 generations. They compared single island, 2 island, 4 island and 8 island configurations. Their experiments have shown that, island model surpassed sequential GA, both in terms of solution quality and completion time. Authors interpreted this as, vast differences are due to preservation of the diversity with the island model (Perreira & Lapa, 2003).

In 2004, Tang et al. studied the effects of migration topology in island model parallel GA, using quadratic assignment problem. They considered two migration models, unidirectional ring migration and a randomized stepping stone model, where recipient islands and receiving individuals are chosen at random. To ensure a fair comparison, they fixed all other parameters as, population size being 240, crossover probability is set to 0.8, mutation probability to 0.05, migration interval is set to every 10 generations, and migration policy is set to best individual replaces worst of the recipient population. They run their algorithms with 3, 4, 6, 8, 10 islands on Skorin-Kapov 100b and Taillard

100b instances. They found out that, unidirectional ring migration schema performed better than the randomized-stepping stone model in general (Tang, Lim, Ong, & Er, 2004)ç

In 2010, Ruciński et al. studied the effects of migration topology on an island model differential evolution algorithm using several continuous functions like Rastrigin, Schwefel and Lennard-Jones. They considered 128, 256, 512 and 1024 islands in their experiments. Their experiments were rather extensive; they tested 14 different topologies including various derivatives of unidirectional ring topology, torus, cartwheel and lattice topology, hypercube, broadcast and fully connected. Migration interval is fixed in all tests to 100 generations. They used best individuals replace worst in the receiver island as the migration policy. They report that, the impact of the island size varies with topologies, so it is not a primary factor to consider when deciding on a topology. However, when the number of islands is increased, the topology becomes more important in order to utilize additional islands. They concluded that, ring migration variants perform better than other kind of topologies, because spread of information is faster (Rucinski, Izzo, & Biscani, 2010).

In 2015, Liu and Wang applied island model parallel GA to generalized assignment problem. In this problem, there are N bins with capacity C and M items. The task is to find an assignment of items into bins in such a way that, no bins exceed their capacity and total cost of assignment is minimized. Authors decided to implement an asynchronous migration methodology in order to reduce communication delay. In their algorithm, each island starts with a randomized population, and islands are connected with 2D toroidal grid topology. In this topology, each island has 4 neighbors. The number of islands are decided in runtime depending on the number of available processors. Each island holds an import FIFO queue. This queue considers two criteria: more elite and more diverse solutions have higher chances of being selected. Rate of immigration and immigration interval are constant and parameterized. The authors considered two types of processor scalabilities when benchmarking the algorithm. Strong scaling tests included more CPU cores and are designed to test the speedup, while weak scaling is used to measure solution quality. Authors executed the tests on supercomputers. Strong scaling results showed that, super-linear speedups are attainable, even with synchronous version. Weak scaling experiments have shown that, number of islands utilized is directly related to solution quality, while they have also

shown that, asynchronous version of algorithm performed better than synchronous version (Liu & Wang, 2015).

In 2016, Kurdi applied an island model GA to job shop scheduling problem. In this problem, there are n machines and m jobs. Goal is to find a schedule to assign all the tasks to the machines in such a way that each machine can process only one job at a time and each job should be processed on each machine. The author employed different mutation models on each of the islands to provide better diversification and provide better exploration. The author employed a derivation of 2D unidirectional ring migration. Migration only happens when algorithm cannot find an improvement in all islands for 10 generations. For the migration policy, the author selected migrate worst solution policy. Author also benchmarked his algorithm against different migration policies, one being replace best and other replace random. He concluded that his algorithm compared better against both of them (Kurdi, 2016).

Other than coarse-grained island model and parallelization of time consuming tasks, called master slave model, there is also a third type of parallelization proposed in literature: fine-grained parallel GAs.

Perhaps, the earliest example in literature is Robertson's work from 1987. According to Cantu – Paz (Cantú-Paz, 1998), his algorithm parallelized selection of parents, mating and crossover sections using a massively parallel computer CM-1 which has 16,000 processors (Robertson, 1987).

In 1989, Gorgos – Schleuter (Gorges - Schleuter, 1989) and her colleague Mühlenbein (Mühlbein, 1989) developed a massively parallel fine-grain model called ASPARAGOS which works on massively parallel computers and utilizes an individual based multipopulational structure what is known as cellular model nowadays. In both articles, it is called as the neighborhood model. In their model, neighborhood is defined on a ladder-like two-dimensional grid where each individual occupied a grid point. Mating and selection was done asynchronously between immediate neighbors. They also implemented a hill climbing local search strategy to prevent population from falling into premature convergence. Mühlenbein applied ASPARAGOS to quadratic assignment and traveling salesman problems, and successfully obtained optimum results for Nugent30 and Steinberg36 QAP instances, as well as 40 city TSP and 50 city TSP instances (Mühlbein, 1989). Gorgos-Schleuter applied it to TSP and tested

problem parameters. She concluded both mutation and migration create diversity within population. However, migration is better at creating variance, because these variations are useful compared to random variations achieved with mutation. She found out that, the most important parameters were selection strategy and population sizes. She also found out ASPARAGOS found better solution than simulated annealing and was faster than k-opt local search strategy (Gorges - Schleuter, 1989).

In 1994, Shapiro and Navetta applied massively parallel neighborhood model algorithm to predict RNA secondary structure. They run their algorithm on MP2 supercomputer with 16,384 processors. They utilized a grid based topology where every cell is appointed to a single processor. Their algorithm was able to find solutions up to 5% of optimal solution (Shapiro & Navetta, 1994).

In 1998, Folino et al. solved k-satisfiability problem using a cellular genetic algorithm. They utilized a 2-dimensional toroidal grid to define the neighborhood between cells. They tested their algorithm on a Meiko CS-2 SIMD computer. They compared their algorithm with a sequential algorithm proposed in the literature on 3-SAT problems with 64 to 512 variables. They showed that, their algorithm converged significantly faster than its sequential version, allowing it to solve larger problems in the same amount of time (Folino, Pizzuti, & Spezzano, 1998).

In 2008, Steiner et al. applied cellular model for development of lightweight and stable materials (Steiner, Jin, & Sendhoff, 2008). In the same year, Alba and Dorronsoro applied a Cellular GA to solve vehicle routing problem. (Alba & Dorronsoro, 2008)

In 2009, Luque et al. applied a cellular genetic algorithm to various combinatorial optimization problems like MAX-SAT, massively multimodal deceptive problem and p-median problem. They used a cluster of computers for their experiment. They found out that cellular GA outperformed island model in these experiments (Luque, Alba, & Dorronsoro, 2009).

CHAPTER 4

AN ISLAND MODEL PARALLEL GA FOR QAP

4.1. Formulation and Solution Representation

According to the recent survey on QAP (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007), there are 5 different formulations proposed in the literature: integer linear programming formulations, Mixed Integer Linear Programming (MILP) formulations, permutation based formulations, trace formulation and graph formulation, while, the most popular formulation in literature is permutation based.

This is due to the simplicity of showing a solution in terms of a permutation and for a genetic algorithm such solutions are easy to manipulate and use existing and proven to be effective genetic operators like Partially Mapped Crossover (PMX) and List Order Mutation. In this kind of formulation, locations are defined with the order in the permutation, while facilities are shown with unique numbers between 1 and n . Each facility's position within the permutation defines its location. For instance, if size of the problem is 5, the permutation 53412 defines a solution in which facility 5 is assigned to location 0, facility 3 is assigned to location 1, facility 4 is assigned to location 2, facility 1 is assigned to location 3 and facility 2 is assigned to location 2.

The optimization objective is minimizing total sum of flow between n facilities and the same number of locations, therefore, a fitness formulation can be defined as follows:

$$\min p \in S_n \sum_{i=1}^n \sum_{j=1}^n A_{ij} \times B_{p_i p_j}$$

where S_n is all possible permutations with size n , A_{ij} is the distance between locations i and j , $B_{p_i p_j}$ is the flow of demand between facilities p_i and p_j . In this implementation, A and B matrices can be symmetrical or asymmetrical depending on the problem instance, algorithm will work in either cases.

4.2. Implementation of The Algorithm

The primary aim of this thesis is measuring the effects of parallelization of a serial genetic algorithm designed for solving QAP. The secondary objective is finding the

optimum algorithm configuration when parallelizing the GA. This study aims to answer following questions:

1. How many islands (processors/threads) gives better performance? How many islands are ideal?
2. Will there ever be a situation where increasing number of islands (threads) actually start decreasing solution quality? (Is there a critical-mass situation?)
3. Does period of migration (an epoch) will affect the outcome of algorithm? If so, what is the ideal setting for period migration?
4. How does number of solutions to be exchanged influence solution quality? What is the ideal setting for the number of solutions to be exchanged?
5. Which immigration topology is more suitable?
6. How does the direction of migration will affect the algorithm's outcome? Should the exchange between islands be unidirectional or bidirectional?
7. What should be the ideal setting for the number of elites to be transferred to the next generation directly?
8. How will the parallelization affect time of execution for the algorithm without regarding any kind of optimization?

In order to answer how each parameter affects the solution, first we needed to create an ordinary sequential GA without considering any kind of optimization. The model consists of a number of islands, each of them running the same sequential algorithm for a number of iterations, while in certain time intervals (after a number of iterations), randomly selected solutions are exchanged between islands.

We also implemented a synchronized master-slave model for sequential GA to utilize empty processors/threads to gain some wall clock time for the experiments. Fitness calculations of each algorithm will be divided between the unused CPU cores and when fitness calculations are obtained, algorithm will resume.

We also added a mechanism which uses two opt local search with best improvement pivoting rule, which is only applied to elite solutions, which will improve the best possible solution within each of the elite's neighborhood. We implemented a generation based genetic algorithm with elitism.

Our sequential algorithm starts with randomly created permutations of size n , where n is the problem size. In each iteration, algorithm will first calculate fitness value of each solution within population, order the solutions within the population based on their fitness values, preserve the best m solutions as population elites and select of $n - m$ solutions among the rest using tournament selection methodology. Main reason for selecting this methodology is that, it is proven to be the effective in QAP or TSP like permutation based problems based on our experiences and other authors in the literature such as Tosun et al. pointed out in a similar work (Tosun, Dokeroglu, & Cosar, 2013).

After the selection process, we use a Partially Mapped Crossover (PMX) as the crossover operator, because it is a reasonably well operator when dealing with permutation based problems like TSP and QAP which is also shown by Tosun et al. (Tosun, Dokeroglu, & Cosar, 2013).

After crossover, each child will undergo two kinds of small mutations to preserve diversity and avoid premature convergence problem. We implemented swap mutation and inversion mutations, while, rest of the population undergoes crossover and mutation, elite solutions also undergo a two-opt local search with best improvement to find best possible solutions within that portion of search space. At the end of each iteration, children and elites are combined and number of individuals within population remains constant.

After a certain number of iterations, algorithm starts a migration process. In this step, a group of randomly chosen solutions are migrated according to the utilized migration topology. In our implementation, we used ring migration model, because it is the easiest schema to implement.

In the final step, before moving to next iteration, algorithm will check for the termination criteria and if the criteria is satisfied, algorithm will terminate and fittest solution within the global population will be returned as a result. Termination criteria is described in detail in the next section.

Algorithm 1 Island Model - Mater Slave Hybrid GA

```
1: procedure GA
2:    $NrOfIslands = N$ 
3:    $StoppingCondition = instanceSize * instanceSize * 20,000$ 
4:    $MigrationPeriod = X$ 
5:    $NumberOfGenerations = 0$ 
6:   for each island do
7:     Create a Random subpopulation of size n
8:   while Number of Total Fitness Evaluations  $\leq$   $StoppingCondition$  do
9:      $NumberOfGenerations = NumberOfGenerations + 1$ 
10:    for each island do
11:      Calculate Fitness Values
12:      Reorder Subpopulation based on Fitness Value
13:      Select m number of elites
14:      Select n - m solutions with Tournament Selection
15:      Local Search m Elites with 2Opt Search
16:      PMX Crossover n-m solutions
17:      Mutate n-m solutions
18:      Combine Elites Into New Population
19:    if  $NumberOfGenerations \% MigrationPeriod = 0$  then
20:      Ring Migration
21:  Return BestFoundSolution
```

Figure 4.1. Island GA Pseudocode

A simple experiment has been designed to decide on cross-over and mutation rate parameters. 2 different GA configurations are created; one with high crossover and mutation rates and the other with low crossover and mutation rates. The experiment has been repeated 10 times for both configurations and minimum, maximum, mean and trimmed mean values are calculated. The results are shown in Table 4.1. High cross-over and mutation rate is more successful, so the next experiments will use these rates.

Table 4.1. Parallel Island GA Parameters Initial Experiment

Configuration	Data Instance	RPC Min	RPC Mean	RPC Trimmed Mean	RPC Max
High Cross-over (100%) & Mutation Rate (50%)	tai30a	2.80	3.46	3.41	4.51
	tai40a	3.07	3.73	3.70	4.64
	tai50a	3.00	3.65	3.66	4.28
	tai60a	3.16	3.47	3.46	3.88
	Average		3.01	3.58	3.56
Low Cross-over (60%) &	tai30a	4.06	5.19	5.16	6.59
	tai40a	4.17	5.32	5.36	6.14

Mutation Rate (5%)	tai50a	4.59	5.28	5.26	6.14
	tai60a	3.94	4.71	4.71	5.43
	Average	4.19	5.13	5.12	6.07

After initial experiments, the following parameters and operators shown in Table 4.1 are selected. Since the focus of this work is examining the effects of hyper parameters for island model GA, we did not consider using other operators and optimizing these parameters further. Testing sequential algorithm in terms of solution quality and experimenting with the parameters and operators is left as a future work.

Table 4.2. Sequential GA Parameters

Crossover Operator	PMX crossover with Probability of 100%
Mutation Operator – 1	Swap Mutation with a probability of 50%.
Mutation Operator – 2	Inversion with Probability of 50%
Parent selection methodology	Tournament selection with size 2 and with a probability of 75% for selecting the better individual

4.3. Algorithm Termination

This work aims to investigate the effect of parallelization of a sequential algorithm, primarily in terms of solution quality, as well as measuring the effects of different parameters such as number of islands, immigration interval, immigrant count, immigration topology, number of elites. To make a fair comparison, we have decided to limit the total number of fitness evaluations by a constant value. By trial and error methodology, we came up with the formula below:

$$\text{Total Fitness Evaluations} = \text{instance size} \times \text{instance size} \times 20,000.$$

Each run of the algorithm is allowed to do this many calculation exactly, and after that, the algorithm terminates and reports the best found individual long with its fitness value, as the final result.

4.4. Coding and Benchmarking the Algorithm

We implemented the algorithm using Java programming language on Windows platform. While coding our project, we used Daniel Dyer's Watchmaker Evolutionary Computing Framework as basis for our project (Dyer, 2013).

Since genetic algorithms are stochastic, they heavily rely on randomization, therefore, we have chosen Java implementation of Xoroshiro128+ algorithm by Tommy Ettinger (Ettinger, 2017) as the random number generator. According to a comparison made by Tobias Ibounig on 2894680 samples (Ibounig, 2016), this algorithm performs 58% better than Java default random number generator in terms of runtime.

In order to ensure fairness, we used same seeds on all of our experiments. Each of the tests applied for different configurations consists of 10 independent runs, except first 2 preliminary experiments. All of the ten independent runs are initialized with seeds between 100 and 109 for each different parameter combination.

Like most of the work we come across in literature, we decided to focus on solving QAPLIB instances. As we mentioned in Chapter 1, these instances can be categorized into four main groups, with uniform - randomly generated instances are being the most difficult ones to solve. This group is a good fit to measure our algorithm's initial performance and compare to other state of the art algorithms. Therefore, we decided to start with Taillard 'a' instances, specifically 'a' instances with sizes between 30 and 80. In addition, none of these instances have been solved to optimality yet, so it can be concluded that, they are among the most difficult QAP instances.

We performed our test on a VMWare virtual machine, running on a server having Xeon E5-2660 2.20 Ghz. Processor. Our virtual machine has 4 cores & 8 threads, as well as 8 GBs of RAM reserved. The virtual machine runs Windows Server 2012 R2 as the operating system with JDK 8 64-bit edition installed.

In order to start our experiments, we decided to run some preliminary tests to determine ideal conditions to benchmark our algorithm. We decided to alternate different parameter configurations and observe their results. Therefore, we prepared an extensive test which includes population size, the number of islands, the number of immigrants per island, immigration period and number of elites per island as alternating parameters. We also decided to use single-directional ring migration as

migration topology, since it is the most commonly preferred and simplest to implement. Table 4.2 lists the parameters and their values considered in the experiments.

Table 4.3. Parallel Island GA Preliminary Test – 1 Parameters

Parameter	Value
Population sizes	50 – 100 – 150
Number of Islands	2 – 4 – 8 – 12
Number of Immigrants	5 – 10
Migration Period (Iterations)	10 – 20
Elite Count	1 – 2

We decided to conduct these tests on Taillard 30a data instance. Test included 96 different configurations and 5 independent runs are made for each configuration using seeds 100-104, which makes 480 runs in total. RPC is computed for each configuration and 5 RPCs are averaged for each configuration. Most successful 13 configurations of the preliminary test is given in Table 4.3.

Table 4.4. Parallel Island GA Preliminary Test-1, Top 13 Most Successful Configurations out of 96, based on average RPC

Population Size	Number of Islands	Number of Immigrants	Immigration Period	Elite Count	Migration Topology	Average RPC (%) per 5 Run
50	12	10	20	1	Unidirectional Ring Migration	3.31
50	8	10	10	1		3.41
100	8	10	10	2		3.42
50	8	10	10	2		3.42

50	12	10	10	2		3.58
50	12	10	10	1		3.58
150	12	10	10	2		3.59
100	12	10	20	2		3.60
50	12	5	20	1		3.64
50	8	5	20	1		3.66
50	8	10	20	2		3.67
100	8	5	10	2		3.68
50	8	5	20	2		3.68

Results of the first preliminary experiment has shown that Population size and Number of islands are the most critical factors in algorithm's success. It seems using lower island population and higher number of islands is more reasonable approach. This experiment has also shown that, immigration period might be yet another parameter worth considering. Effects of number of immigrants and elites cannot be determined based on this experiment.

We decided to run a secondary preliminary experiment based on the obtained results. Our purpose was to measure the effect of population size and number of islands without regarding other parameters. Since first test has shown smaller island populations with larger number of islands are more effective, we decreased population sizes and increased number of islands. Table 4.4 shows the second preliminary test parameters.

Table 4.5. Parallel Island GA Preliminary Test -2 Parameters

Parameter	Value
Population sizes	40 – 50 – 60
Number of Islands	10 – 12 – 14
Number of Immigrants	10
Migration Period (Iterations)	10
Elite Count	2

Second preliminary experiment also confirmed our initial assumption. For Taillard 30a instance and presumably also for the other Taillard ‘a’ instances, low island population and high number of islands yields the better performance. The results of this experiment sorted on decreasing average RPCs is given in Table 4.5.

Table 4.6 Parallel Island GA Preliminary Test -2 Results

Population Size	Number of Islands	Average RPC (%) per 5 Run	Minimum RPC (%) Per 5 Run
40	14	2.997	2.720
40	12	3.292	2.732
40	10	3.508	2.872
50	14	3.531	2.893
50	12	3.575	2.858
50	10	3.684	3.035
60	12	3.687	2.594
60	14	3.691	2.922
60	10	3.769	3.034

In the next experiments, we use a static global population size and vary the number of groups they are divided (Islands / Threads). We decided to use a global population of 600 solutions to measure direct effect of number of islands / threads on solution quality. We prepared yet another experiment with different number of islands, varying between 1 and 75, with 1 being the original sequential algorithm with all of 600 individuals on a single island. In following test, each island will contain (600 / Number of Islands) solutions. Since the population sizes on each island changes along with the number of

islands, we decided to use ratios instead of fixed values for parameters. Table 4.6 shows the parameter values used in this experiment.

Table 4.7 Parallel Island GA Optimizing Number of Islands

Parameter	Value
Number of Immigrants	10 % of Population size per island
Migration Period (Iterations)	5 x Number of Islands
Elite Count	1 % of Population size per island or 1

In the next experiment, we have considered out of total $30 \times 30 \times 20,000 = 18,000,000$ fitness evaluations as the termination limit for tai30a instance. The results are given in Table 4.7

Table 4.8 Number of Islands Experiment Results with Tai30a data set

Population Size	Number of Islands	Minimum RPC Per 10 run	Mean RPC per 10 Run	Trimmed Mean RPC Per 10 Run	Maximum RPC for 10 runs
600	1	11.47	12.64	12.70	13.32
300	2	3.65	4.34	4.28	5.47
150	4	3.37	4.10	4.13	4.56
75	8	3.24	4.11	4.11	4.98
60	10	2.89	4.09	4.07	5.38
50	12	3.28	4.07	4.01	5.34
40	15	2.18	3.57	3.67	4.18
30	20	2.21	3.57	3.61	4.60
20	30	3.31	4.08	4.07	4.91
15	40	2.47	4.03	4.05	5.39
12	50	2.39	3.75	3.79	4.78
10	60	3.99	4.46	4.44	5.14
8	75	4.06	5.08	4.98	6.94

Mean RPC values versus number of islands plot is shown in Figure 4.1. We can see a sharp increase in solution quality when 2 islands are used, due to the benefit gained from two diverse populations rather than a single one. Solution quality of 2 islands and others are almost 3 times better than sequential algorithm counterpart. This evidently

shows the power of island model GA compared to classical GA. For other results, while 15 - 20 island (Threads/Processors) yields the best results in terms of solution quality, we can see a gradual decrease until 40 - 50 Islands. After that point, algorithm performance starts to drop, presumably because subpopulation amount is not enough to create a diverse range population, thus hindering evolution process. We call this point as “Critical Mass”, where any further parallelization will make solution quality worse.

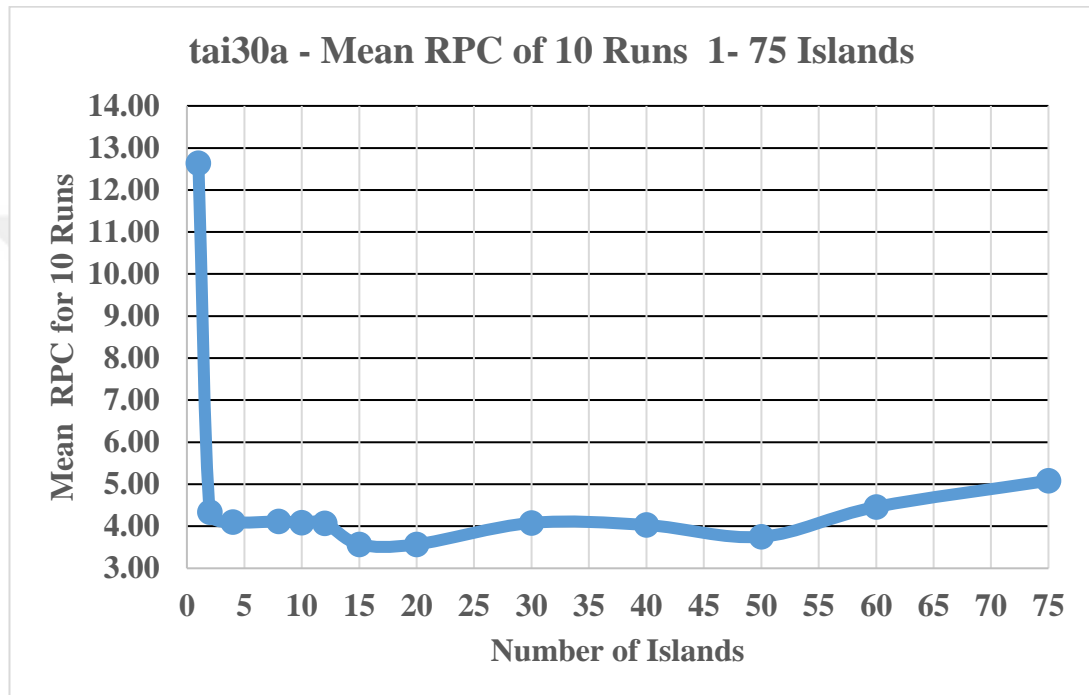


Figure 4.2. Number of islands versus mean RPC graph for Tai30a instance

For tai40a instance, we have considered out of $40 \times 40 \times 20,000 = 32,000,000$ fitness calculations for each run. Detailed results are given in Table 4.8

Table 4.9 Number of Islands Experiment Results with Tai40a data set

Population Size	Number of Islands	Minimum RPC Per 10 run	Mean RPC per 10 Run	Trimmed Mean RPC Per 10 Run	Maximum RPC for 10 runs
600	1	13.63	13.97	13.98	14.23
300	2	2.03	4.30	4.45	5.34
150	4	2.03	4.64	4.77	6.23
75	8	2.03	4.50	4.63	5.89
60	10	2.03	4.43	4.55	5.84

50	12	2.03	3.81	3.91	4.78
40	15	3.44	4.25	4.21	5.29
30	20	2.99	4.15	4.18	5.06
20	30	3.82	4.51	4.50	5.33
15	40	2.86	4.37	4.34	6.14
12	50	3.11	3.91	3.90	4.77
10	60	2.82	4.82	4.93	5.93
8	75	4.20	5.16	5.17	6.08

Just as in tai30a instance set, solution quality of island model GA is almost 3 times better with minimum number of islands (2). It also shows us, while 12 islands performed best, it may be coincidental because solution quality decreases gradually up to 30 islands and after that point, results start to get better and better until 50 islands, which is the second-best point after 12 islands. After 50 islands, solution quality starts to drop slowly again. Therefore, we concluded that 50 islands is the also the critical mass point for tai40 instance as well. Mean RPC values versus number of islands plot is shown in Figure 4.2.

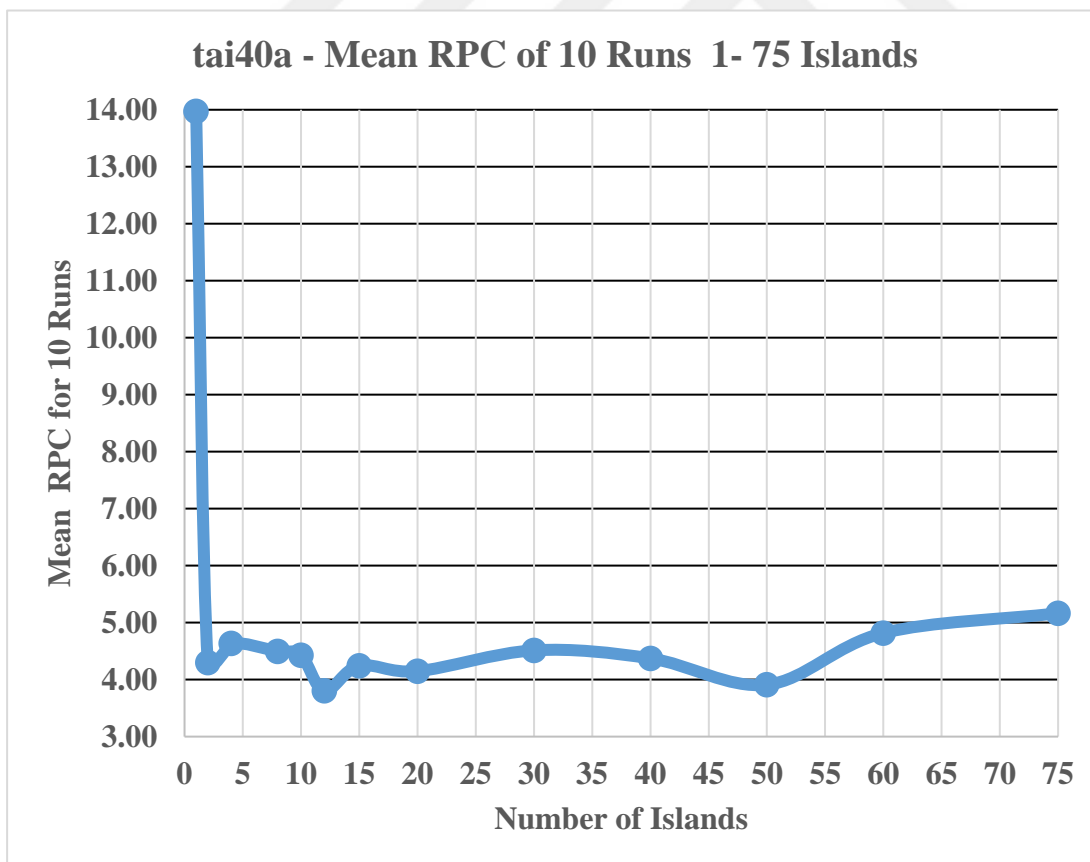


Figure 4.3. Number of islands versus mean RPC graph for Tai40a instance

For tai50a instance, we have considered $50 \times 50 \times 20,000 = 50,000,000$ fitness calculations. Table 4.9 shows the detailed results. Just as previous test instances, 20 islands seem to produce best average results, however, solution quality starts to drop after 20 islands, and stands low until 50 islands, which is also the second-best point in terms of average solution quality. After 50 islands, we see same solution quality drop presumably due to insufficient island population hindering evolution. Figure 4.3 shows the plot of number of islands versus mean RPC graph for Tai40a instance.

Table 4.10 Number of Islands Experiment Results with Tai50a data set

Population Size	Number of Islands	Minimum RPC Per 10 runs	Mean RPC per 10 runs	Trimmed Mean RPC Per 10 runs	Maximum RPC for 10 runs
600	1	13.57	14.08	14.11	14.33
300	2	3.91	4.63	4.65	5.12
150	4	3.91	4.43	4.43	4.99
75	8	3.25	4.32	4.33	5.23
60	10	3.56	4.30	4.33	4.83
50	12	3.11	4.28	4.32	5.12
40	15	3.60	4.32	4.28	5.39
30	20	3.14	3.95	3.98	4.54
20	30	3.90	4.44	4.41	5.20
15	40	3.93	4.53	4.50	5.41
12	50	3.46	4.22	4.22	4.91
10	60	3.53	4.54	4.58	5.25
8	75	4.24	5.01	5.00	5.84

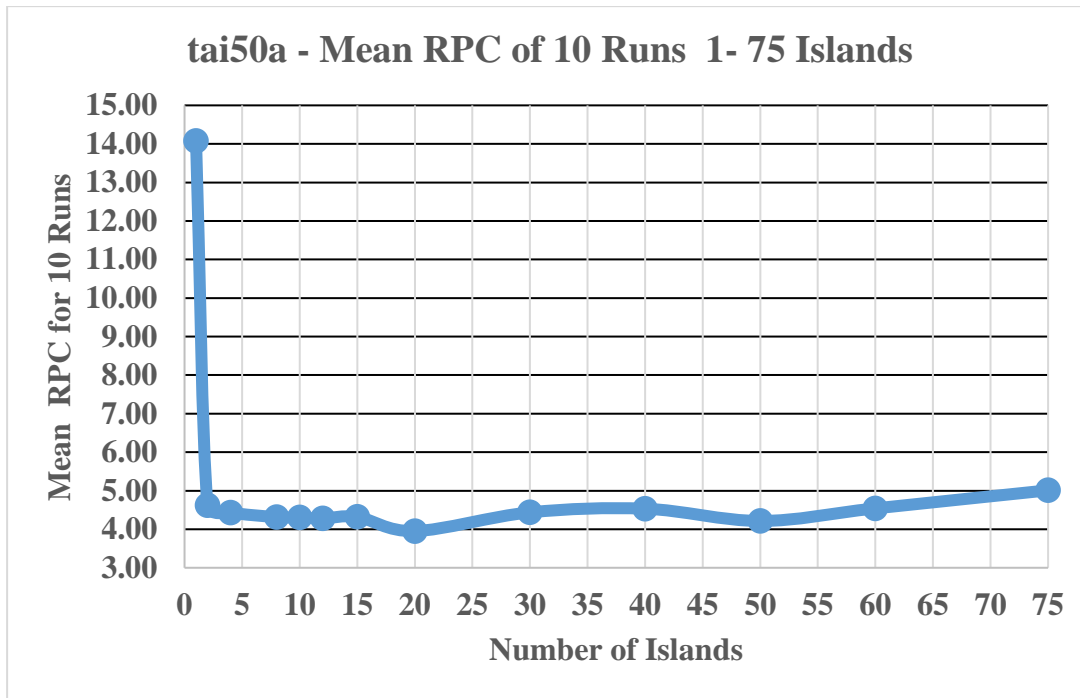


Figure 4.4. Number of islands versus mean RPC graph for Tai50a instance

For tai60a problem instance, we have considered $60 \times 60 \times 20,000 = 72,000,000$ fitness calculations. We can see the same almost 3 times increase in terms of solution quality between sequential and island model GAs. In terms of average solution quality, 50 islands seem to produce best average solution quality out of 10 runs. After 50 islands, solution quality starts to drop. Table 4.10 gives the detailed RPC values and Figure 4.4 shows the number of islands versus mean RPC plot.

Table 4.11 Number of Islands Experiment Results with Tai60a data set

Population Size	Number of Islands	Minimum RPC Per 10 Runs	Mean RPC per 10 Runs	Trimmed Mean RPC Per 10 Runs	Maximum RPC for 10 Runs
600	1	13.13	13.72	13.76	13.92
300	2	3.34	4.32	4.36	4.99
150	4	3.34	4.34	4.37	5.04
75	8	3.34	4.14	4.06	5.57
60	10	3.62	4.14	4.10	4.95
50	12	2.76	4.26	4.28	5.56
40	15	3.76	4.11	4.05	4.94
30	20	3.28	3.96	4.00	4.38
20	30	3.26	4.03	3.99	5.06
15	40	3.56	3.97	3.93	4.72
12	50	3.18	3.85	3.89	4.19
10	60	2.55	4.02	4.12	4.71
8	75	3.29	4.25	4.26	5.14

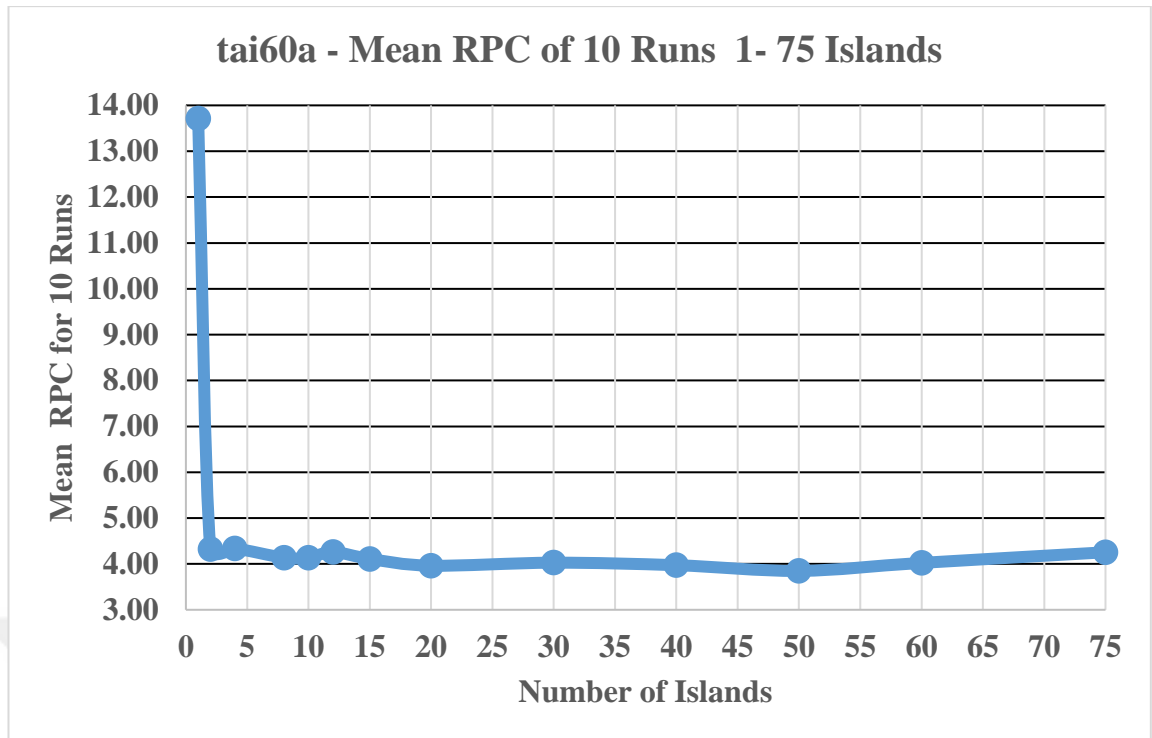


Figure 4.5. Number of islands versus mean RPC graph for Tai60a instance

For tai80a instance, we have considered $80 \times 80 \times 20,000 = 128,000,000$ fitness calculations. As with all the previous cases, island model is 3 times more effective than its sequential counterpart in term of solution quality. In terms of average solution quality, 20 islands configuration has a slight edge over 50 islands, but they are close. After 50 islands, same scenario happens and solution quality starts dropping. Table 4.11 gives the detailed RPC values and Figure 4.5 shows the number of islands versus mean RPC plot.

Table 4.12 Number of Islands Experiment Results with Tai80a data set

Population Size	Number of Islands	Minimum RPC Per 10 Runs	Mean RPC per 10 Runs	Trimmed Mean RPC Per 10 Runs	Maximum RPC for 10 Runs
600	1	12.16	12.47	12.47	12.71
300	2	3.04	3.70	3.71	4.32
150	4	3.15	3.82	3.74	5.15
75	8	3.18	3.77	3.78	4.32
60	10	3.15	3.65	3.65	4.21
50	12	3.27	3.67	3.65	4.21
40	15	2.93	3.53	3.55	3.97
30	20	2.78	3.56	3.58	4.24
20	30	3.42	3.94	3.93	4.52
15	40	3.24	3.67	3.64	4.34

12	50	2.95	3.66	3.67	4.30
10	60	3.13	3.87	3.86	4.72
8	75	3.82	4.31	4.25	5.29

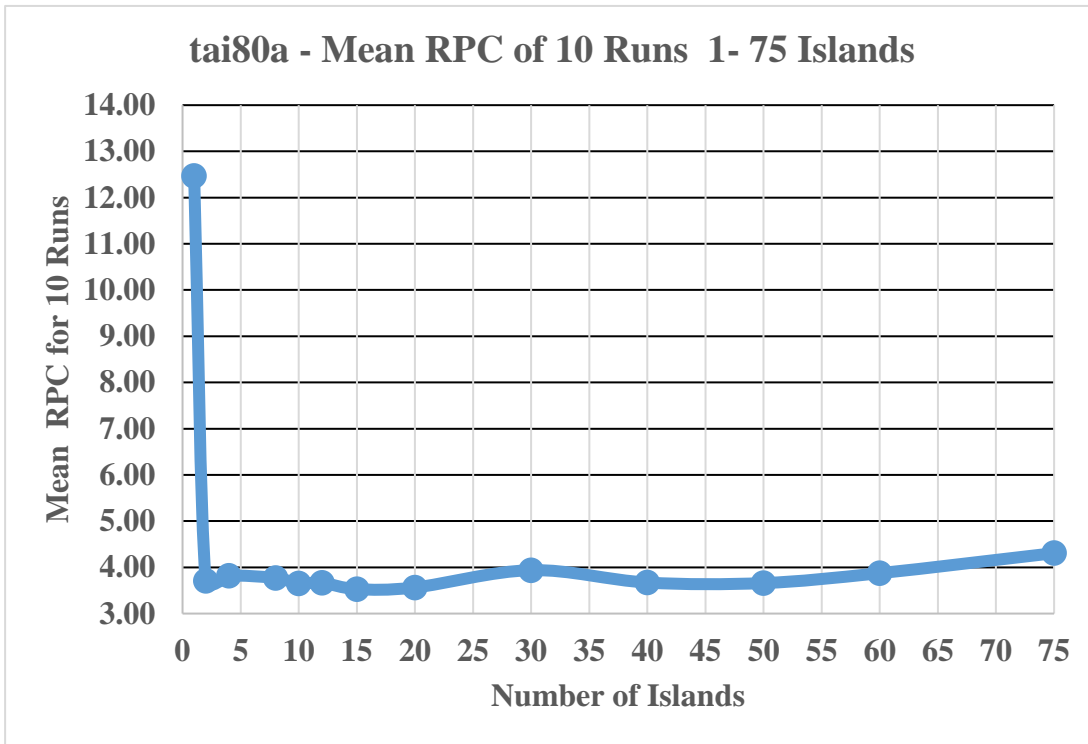


Figure 4.6. Number of islands versus mean RPC graph for Tai80a instance

Third experiment showed us that island model is able to produce 3 times better results than its sequential counterpart. If we consider the ideal number of islands, while 20 islands can be considered as a good choice for Taillard ‘a’ data set, there are improvements until 50 islands and 50 islands can be considered as critical mass point for this data set using 600 global population.

We have also examined 1 - 75 island configurations in terms of speedup over their sequential counterpart. Table 4.12 shows the parallelization efficiencies for different numbers of islands, as well as single island & master-slave model (1-island). Each of the configurations shown in the table utilizes all the processing units within the computer (4 cores and 8 threads). When implementing the algorithm, we have not considered optimizing completion time, since our main focus was the improvement in terms of solution quality. Despite this, we can see super-linear speedups both in 1 island master - slave model and in our island & master - slave hybrid algorithm. We can see that although the parallelization efficiency is still super-linear, it is a little low for 2 - islands configuration, compared to other cases. After 4 islands, rate of

improvement starts to increase until 50 islands, peaking at 40-islands configuration with an average value of 1.36. After 50 islands, number of threads utilized by the algorithm is considerably over the limits of physical hardware causing bottlenecks, and rate of improvement starts to decrease again. Note that, in our algorithm, migration model is synchronous which means there may be idle times when some processing units wait for the others while the algorithm runs. Converting algorithm to asynchronous might also improve run time. Figure 4.6 also shows the parallelization efficiency graphically.

Table 4.13 Parallelization Efficiency (%) on Taillard samples.

Parallelization Efficiency in Taillard Samples compared to Seq. Alg.													
# of Islands	1 (Master Slave)	2	4	8	10	12	15	20	30	40	50	60	75
Taillard30a	1.19	1.11	1.24	1.27	1.35	1.37	1.33	1.44	1.48	1.48	1.25	1.16	1.14
Taillard40a	1.25	1.04	1.21	1.25	1.30	1.26	1.31	1.31	1.22	1.32	1.10	1.01	0.99
Taillard50a	1.27	0.98	1.17	1.27	1.29	1.28	1.27	1.29	1.23	1.31	1.08	0.99	0.95
Taillard60a	1.32	0.96	1.13	1.30	1.30	1.27	1.28	1.29	1.27	1.33	1.08	1.00	0.97
Average	1.26	1.02	1.19	1.27	1.31	1.29	1.30	1.33	1.30	1.36	1.12	1.04	1.01

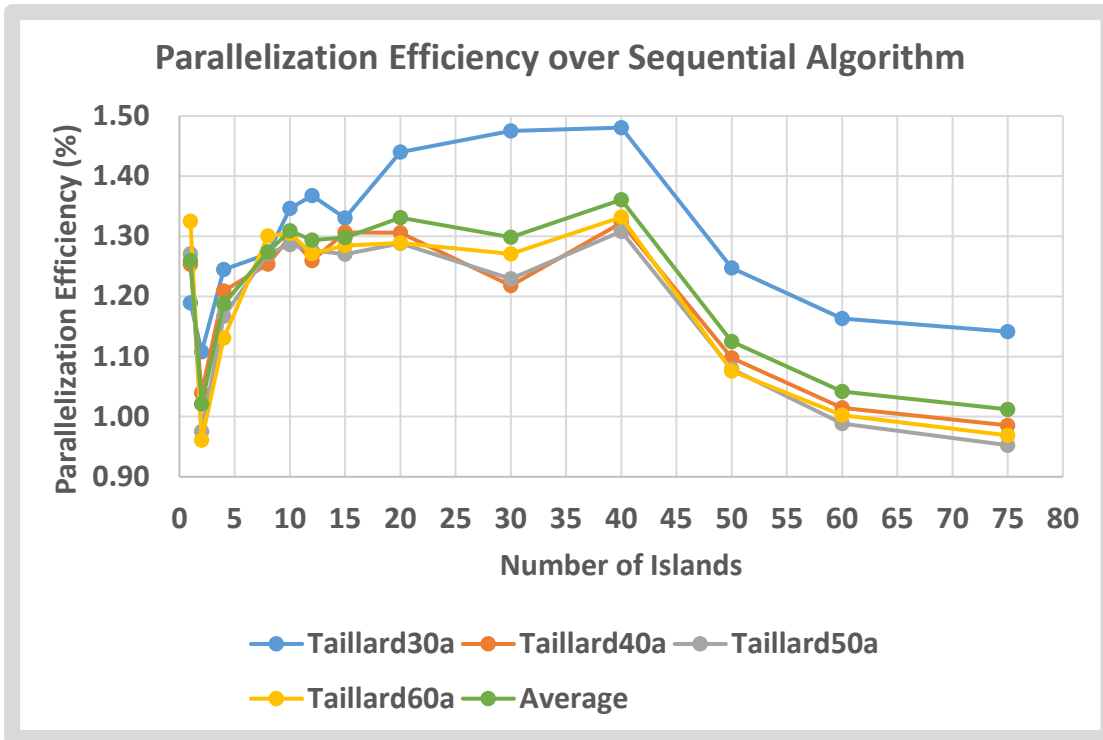


Figure 4.7. Parallelization efficiency of Island Model over sequential Algorithm

After the third experiment, we decided that best configuration we can use on Taillard problem instances are 50 islands and 12 individuals for each island, since, it would guarantee to give nearly best results for each instance and still has a fairly good speedup. So far in all tests, we tested using unidirectional ring migration model. We also wanted to see if changing migration topology might make things different. Therefore, we utilized a secondary model that uses a bidirectional ring model and compared it with unidirectional counterparts for 30, 40 and 50 island configurations. In classical ring migration model, every island sends m randomly chosen solutions to the next island, starting from the first island. Island at the end sends to the first island, thus, forming a ring. In bidirectional model, starting from first island, every island sends $m/2$ randomly chosen solutions to the next island, while other $m/2$ solutions are sent to the previous island. If m is odd, $(m + 1)/2$ solutions are sent to the next island, while, $m - ((m + 1)/2)$ solutions are sent to the previous island. First island's previous island is considered to be last island and last island's next is considered as first island, therefore forming a ring.

The results for unidirectional and bidirectional ring migration topologies for different Taillard instances are shown in Table 4.13 to Table 4.17. In the fourth experiment, we observed that, the results do not show significant differences, so we cannot conclude

which topology is better. Their performances are very close for 30a, 40a and 50a instances. For Taillard60a instance, unidirectional migration performs better and if we look at Taillard80a data set, bidirectional migration seems better. However, given that, we have not done a statistical analysis on results, we cannot conclude that one is better than the other for Taillard – a instances. We think these inconclusive results may be caused by random-migration policy. In order to confirm this theory, additional experiments need to be conducted by utilizing other fitness-based migration policies. For the sake of simplicity, we continued with unidirectional ring migration model.

Table 4.14 Unidirectional – Bidirectional Ring Migration Comparison on Tai30a

Taillard 30a			
Population Size	Number of Islands	Migration Topology	Mean RPC per 10 Run
30	20	Single Direction	3.57
30	20	Bidirectional	3.70
20	30	Bidirectional	3.88
20	30	Single Direction	4.08
15	40	Bidirectional	3.84
15	40	Single Direction	4.03
12	50	Single Direction	3.75
12	50	Bidirectional	3.81

Table 4.15 Unidirectional – Bidirectional Ring Migration Comparison on Tai40a

Taillard 40a			
Population Size	Number of Islands	Migration Topology	Mean RPC per 10 Run
30	20	Single Direction	4.15
30	20	Bidirectional	4.26
20	30	Bidirectional	4.30
20	30	Single Direction	4.51
15	40	Bidirectional	4.12
15	40	Single Direction	4.37

12	50	Single Direction	3.91
12	50	Bidirectional	3.79

Table 4.16 Unidirectional - Bidirectional Ring Migration Comparison on Tai50a

Taillard 50a			
Population Size	Number of Islands	Migration Topology	Mean RPC per 10 Run
30	20	Single Direction	3.95
30	20	Bidirectional	4.00
20	30	Bidirectional	4.25
20	30	Single Direction	4.44
15	40	Bidirectional	4.45
15	40	Single Direction	4.53
12	50	Single Direction	4.22
12	50	Bidirectional	4.29

Table 4.17 Unidirectional - Bidirectional Ring Migration Comparison on Tai60a

Taillard 60a			
Population Size	Number of Islands	Migration Topology	Mean RPC per 10 Run
30	20	Single Direction	3.96
30	20	Bidirectional	4.02
20	30	Bidirectional	4.00
20	30	Single Direction	4.03
15	40	Single Direction	3.97
15	40	Bidirectional	4.06
12	50	Single Direction	3.85
12	50	Bidirectional	3.94

Table 4.18 Unidirectional - Bidirectional Ring Migration Comparison on Tai80a

Taillard 80a			
Population Size	Number of Islands	Migration Topology	Mean RPC per 10 Run
30	20	Bidirectional	3.50
30	20	Single Direction	3.56
20	30	Bidirectional	3.89

20	30	Single Direction	3.94
15	40	Single Direction	3.67
15	40	Bidirectional	3.69
12	50	Bidirectional	3.57
12	50	Single Direction	3.66

After having decided number of islands and migration type, we tried to optimize other parameters. For the experiment 3, we made educated guesses for number of immigrants, migration period (iterations) and elite count parameters. Therefore, in the fifth experiment we tried to optimize these parameters on Taillard 30-60a instances using 40 and 50 island configurations and unidirectional ring migration model. We altered only one parameter at a time to get their best scores. Considered parameters and their values are shown in Table 4.18. In the table, TBT is short for to be tested, while Original is the value used from experiment 3.

Table 4.19 : Optimizing Other Parameters on Taillard a Instances – Possible Configurations

Parameter	Value
Number of Immigrants	10 % of Population size per island (Original)
	20 % of Population size per island (TBT)
	5% of Population size per island (TBT)
Migration Period (Iterations)	3 x Number of Islands (TBT)
	4 x Number of Islands (TBT)
	5 x Number of Islands (Original)
	6 x Number of Islands (TBT)
	7 x Number of Islands (TBT)
Elite Count	1 – Elite per island (Original)
	2 – Elites per island (TBT)

We first started with optimizing number of immigrants parameter. Originally, it was set to 10% which is 2 in an all cases. We tried swapping 4 individuals as well as 1 individual on each of the data sets. The results are shown in Table 4.19. Results clearly indicate swapping 4 elements instead of 2 creates better results, while swapping a single element always led to worse solutions. We did not try increasing this any further, because it may destabilize evolution process within the islands.

Table 4.20 Taillard a Samples Optimizing Number of Immigrants

Data Set	Population Size	Number of Islands	Param. Status	Number of Immigrants	Mean RPC per 10 Run
40 Islands					
Tai30a	15	40	20%	4	3.52
	15	40	Original (10%)	2	4.03
	15	40	5%	1	4.06
50 Islands					
Tai30a	12	50	20%	4	3.48
	12	50	Original (10%)	2	3.75
	12	50	5%	1	4.10
40 Islands					
Tai40a	15	40	20%	4	3.86
	15	40	Original (10%)	2	4.37
	15	40	5%	1	4.50
50 Islands					
Tai40a	12	50	20%	4	3.75
	12	50	Original (10%)	2	3.91
	12	50	5%	1	4.19
40 Islands					
Tai50a	15	40	20%	4	4.12
	15	40	Original (10%)	2	4.53
	15	40	5%	1	5.05
50 Islands					
Tai50a	12	50	20%	4	4.16
	12	50	Original (10%)	2	4.22
	12	50	5%	1	4.44
40 Islands					
Tai60a	15	40	20%	4	3.75
	15	40	Original (10%)	2	3.97
	15	40	5%	1	4.30
50 Islands					
Tai60a	12	50	Original (10%)	2	3.85
	12	50	20%	4	3.88

	12	50	5%	1	4.28
--	----	----	----	---	------

Then, we experimented on immigration period parameters, which was originally set to 5 X Number of Islands in experiment 3. We tried 3, 4, 6 and 7 times the number of islands values for this parameter and compared to the original value. The results are shown in Table 4.20. Our experiments show that, 3X seems better, because, it was superior to the all others, for except Taillard40a – 40 island configuration. Note that, immigration times are given in terms of number of iterations in the table.

Table 4.21 : Taillard a Samples Optimizing Immigration Time (Number of Iterations)

Data Set	Population Size	Number of Islands	Param. Stat.	Immigration Time	Mean RPC per 10 Run
Tai30a	15	40	3X	120	3.83
	15	40	5X (Original)	200	4.03
	15	40	4X	160	4.03
	15	40	7X	280	4.21
	15	40	6X	240	4.38
	12	50	3X	150	3.51
	12	50	4X	200	3.59
	12	50	5X (Original)	250	3.75
	12	50	6X	300	4.24
	12	50	7X	350	4.32
	Tai40a	15	40	3X	120
15		40	4X	160	4.17
15		40	7X	280	4.20
15		40	6X	240	4.23
15		40	5X (Original)	200	4.37
12		50	5X (Original)	250	3.91
12		50	3X	150	4.09
12		50	4X	200	4.17
12		50	7X	350	4.25
12		50	6X	300	4.60
Tai50a		15	40	3X	120
	15	40	6X	240	4.16

	15	40	4X	160	4.19
	15	40	7X	280	4.29
	15	40	5X (Original)	200	4.53
	12	50	3X	150	4.10
	12	50	5X (Original)	250	4.22
	12	50	4X	200	4.26
	12	50	7X	350	4.28
	12	50	6X	300	4.89
Tai60a					
	15	40	3X	120	3.79
	15	40	4X	160	3.89
	15	40	5X (Original)	200	3.97
	15	40	6X	240	4.05
	15	40	7X	280	4.12
	12	50	3X	150	3.85
	12	50	5X (Original)	250	3.85
	12	50	4X	200	3.96
	12	50	7X	350	4.18
	12	50	6X	300	4.49

Last parameter we experimented on was number of elite solutions per island. We experimented with 2% instead of 1%. The results are shown in Table 4.21. Results indicate that, in all cases, %1 is better than %2.

Table 4.22 Taillard a Samples Optimizing Number of Elites per Island

Data Set	Population Size	Number of Islands	Elite Count	Mean RPC per 10 Run
Tai30a	40 Islands			
	15	40	1 (Original)	4.03
	15	40	2	4.35
	50 Islands			
	12	50	1 (Original)	3.75
	12	50	2	4.38
Tai40a	40 Islands			
	15	40	1 (Original)	4.37
	15	40	2	5.00
	50 Islands			
	12	50	1 (Original)	3.91
	12	50	2	4.80
Tai50a	40 Islands			

	15	40	1 (Original)	4.53
	15	40	2	4.96
	50 Islands			
	12	50	1 (Original)	4.22
	12	50	2	4.98
Tai60a	40 Islands			
	15	40	1 (Original)	3.97
	15	40	2	4.91
	50 Islands			
	12	50	1 (Original)	3.85
	12	50	2	4.46

In this thesis, we tried to apply a good parallelization method to a standard sequential genetic algorithm. Our results seem promising. Our work on Taillard data set, which is one of most difficult data sets on QAPLIB, have revealed a good enough island model parallelization method for a standard sequential GA. We are able to obtain 3 times better results with slightly higher time frame. Although speed up optimization is not performed. Our findings indicate that configuration provided in Table 4.22 using standard island model is very efficient.

Table 4.23 : Taillard a Samples Best Parameters found on Taillard a Data Set

Parameter	Value
Global Population Size	600
Island Population Size	12
Number of Islands	50
Number of Immigrants	4 (20% of Population size per island)
Migration Period (Iterations)	150 (3 x Number of Islands)
Elite Count	1 % of Population size per island or 1
Migration Type	Unidirectional Ring Migration
Migration Policy	Immigrants replaced
Immigrant Selection	Randomly from subpopulations.

Next, we used the parameter values given in Table 4.22 to test the algorithm on other Tai ‘a’ instances. Table 4.23 shows the results. The algorithm is executed 10 times on each of the instance using random seeds 100 to 109. We can see that this algorithm is able to achieve 2.8% RPC on smallest instance and 3% on average on Taillard - a data set.

Table 4.24 : Taillard a Instances Optimum Configuration Results

Data Instance	RPC Min	RPC Mean	RPC Trimmed Mean	RPC Max
tai30a	2.798	3.460	3.412	4.506
tai40a	3.073	3.732	3.701	4.635
tai50a	2.995	3.653	3.656	4.283
tai60a	3.163	3.474	3.462	3.875
Average	3.010	3.580	3.560	4.320

Since our results with Taillard a instance set were somewhat conclusive in terms of solution quality, we also decided to test our algorithm on other data sets. As we mentioned before, QAP Instances can be classified roughly in 4 categories. We tested with Uniform Randomly Generated Instance Taillard ‘a’ instance set. Therefore, we decided to test other 3 categories as well. For real-life instances, we decided to use Burkard and Steinberg instance set from QAPLIB. We repeated experiment 3 with same parameters we used on Taillard ‘a’ experiment, namely the effect of number of islands on solution quality.

We used Burkard26a, Burkard26b, Burkard26c, Burkard26d, Steinberg36a, Steinberg36b and Steinberg36c instances. Experiments are run on same machine as before using same seed range. The results are given in Table 4.24. Best configurations in terms of solution quality are highlighted with green.

Table 4.25 : Burkard Instance Set Optimizing Number of Islands

Burkard 26A Results of 10 Runs for each Configuration					
Number of Islands	Population per Island	Minimum RPC %	Mean RPC %	Trimmed Mean RPC %	Maximum RPC %
Single Population	600	1.69	2.31	2.34	2.68
2 - Islands	300	0.11	0.27	0.27	0.44
4 - Islands	150	0.03	0.24	0.22	0.56
8 - Islands	75	0.00	0.18	0.18	0.36
10 - Islands	60	0.09	0.21	0.21	0.34
12 - Islands	50	0.00	0.16	0.15	0.38
15 - Islands	40	0.10	0.18	0.17	0.32
20 - Islands	30	0.03	0.19	0.20	0.27
30 - Islands	20	0.08	0.20	0.19	0.38
40 - Islands	15	0.02	0.15	0.14	0.38
50 - Islands	12	0.00	0.18	0.17	0.48
60 - Islands	10	0.09	0.24	0.23	0.48
75 - Islands	8	0.10	0.29	0.29	0.48
Burkard 26B Results of 10 Runs for each Configuration					
Single Population	600	1.63	2.35	2.38	2.81
2 - Islands	300	0.00	0.22	0.21	0.52
4 - Islands	150	0.00	0.29	0.30	0.51
8 - Islands	75	0.18	0.34	0.32	0.73
10 - Islands	60	0.02	0.23	0.22	0.50
12 - Islands	50	0.17	0.24	0.24	0.38
15 - Islands	40	0.17	0.27	0.25	0.50
20 - Islands	30	0.17	0.26	0.25	0.35
30 - Islands	20	0.17	0.24	0.23	0.38
40 - Islands	15	0.17	0.24	0.22	0.42
50 - Islands	12	0.00	0.24	0.22	0.61
60 - Islands	10	0.17	0.31	0.29	0.61
75 - Islands	8	0.18	0.37	0.35	0.72
Burkard 26C Results of 10 Runs for each Configuration					
Single Population	600	2.04	2.47	2.45	3.09
2 - Islands	300	0.00	0.09	0.08	0.27
4 - Islands	150	0.02	0.20	0.16	0.67
8 - Islands	75	0.00	0.13	0.07	0.77
10 - Islands	60	0.00	0.27	0.25	0.68
12 - Islands	50	0.01	0.08	0.06	0.25
15 - Islands	40	0.03	0.25	0.23	0.66

20 - Islands	30	0.01	0.14	0.10	0.58
30 - Islands	20	0.00	0.11	0.07	0.47
40 - Islands	15	0.02	0.12	0.10	0.37
50 - Islands	12	0.00	0.18	0.15	0.59
60 - Islands	10	0.01	0.29	0.23	0.99
75 - Islands	8	0.02	0.33	0.27	1.11
Burkard 26D Results of 10 Runs for each Configuration					
Single Population	600	2.05	2.58	2.58	3.07
2 - Islands	300	0.01	0.08	0.07	0.31
4 - Islands	150	0.01	0.29	0.25	0.86
8 - Islands	75	0.00	0.17	0.10	0.95
10 - Islands	60	0.00	0.28	0.23	0.95
12 - Islands	50	0.01	0.13	0.08	0.67
15 - Islands	40	0.01	0.13	0.11	0.39
20 - Islands	30	0.00	0.08	0.05	0.37
30 - Islands	20	0.00	0.04	0.03	0.20
40 - Islands	15	0.01	0.10	0.07	0.38
50 - Islands	12	0.00	0.06	0.04	0.23
60 - Islands	10	0.01	0.24	0.16	1.08
75 - Islands	8	0.01	0.36	0.30	1.22

In general, results improve when more islands are used up to 50 islands configuration. Only exception was C instance, in which improvement continued up to 40 islands. Therefore, in general we can conclude that, 50 islands is the critical mass point for the Burkard data set also. After this point, there is a drop in solution quality due to insufficient intra island population size.

Results for Steinberg instances using the same parameter configuration are given in Table 4.25. Again, best configurations for each instance are highlighted as gray.

Table 4.26 : Steinberg Instance Set Optimizing Number of Islands

Steinberg36a Results of 10 Runs for each Configuration					
Number of Islands	Population per Island	Minimum RPC %	Mean RPC %	Trimmed Mean RPC %	Maximum RPC %
Single Population	600	61.08	65.10	65.06	69.41
2 - Islands	300	4.20	7.87	7.85	11.72
4 - Islands	150	5.56	10.39	10.41	15.08
8 - Islands	75	4.89	7.26	7.20	10.08
10 - Islands	60	4.70	8.37	8.15	13.79

12 - Islands	50	4.98	7.52	7.16	12.98
15 - Islands	40	3.91	7.52	7.02	15.16
20 - Islands	30	4.54	7.89	7.66	13.10
30 - Islands	20	5.10	7.69	7.34	13.10
40 - Islands	15	5.50	8.31	7.86	14.68
50 - Islands	12	4.30	7.60	6.92	16.31
60 - Islands	10	3.84	8.09	7.90	13.88
75 - Islands	8	5.29	9.02	8.70	15.24
Steinberg36b Results of 10 Runs for each Configuration					
Number of Islands	Population per Island	Minimum RPC %	Mean RPC %	Trimmed Mean RPC %	Maximum RPC %
Single Population	600	156.62	170.34	170.45	183.20
2 - Islands	300	4.19	14.87	14.79	26.22
4 - Islands	150	7.77	14.05	13.23	26.87
8 - Islands	75	5.01	14.79	14.65	25.69
10 - Islands	60	3.96	14.58	14.80	23.42
12 - Islands	50	2.76	11.51	11.64	19.20
15 - Islands	40	5.64	15.93	15.85	26.87
20 - Islands	30	3.90	10.74	10.86	16.59
30 - Islands	20	6.79	14.52	13.91	27.21
40 - Islands	15	8.37	12.91	13.16	15.51
50 - Islands	12	5.79	10.73	10.40	18.33
60 - Islands	10	11.31	15.77	15.77	20.28
75 - Islands	8	8.16	20.55	18.54	49.00
Steinberg36c Results of 10 Runs for each Configuration					
Number of Islands	Population per Island	Minimum RPC %	Mean RPC %	Trimmed Mean RPC %	Maximum RPC %
Single Population	600	54.16	57.40	57.32	61.28
2 - Islands	300	3.25	5.09	5.02	7.51
4 - Islands	150	4.38	8.67	8.47	14.60
8 - Islands	75	3.01	7.28	7.09	13.09
10 - Islands	60	2.70	6.45	6.22	12.06
12 - Islands	50	2.70	6.88	6.84	11.38
15 - Islands	40	3.86	8.63	8.40	15.23
20 - Islands	30	2.94	6.37	6.16	11.56
30 - Islands	20	2.12	5.20	5.28	7.67
40 - Islands	15	4.42	7.14	6.66	13.72
50 - Islands	12	2.76	6.11	6.00	10.39
60 - Islands	10	2.82	8.30	8.20	14.60
75 - Islands	8	4.34	8.50	8.26	14.60

Just like the previous data sets, we can clearly see that 50 islands configuration is one of three best found configurations in each test. Therefore, we may say that 50 islands is also best selection for the Steinberg instances in general. Also, we notice that, parallelization also works like a charm here, creating 12 to 15 times better results even in 2 island configurations.

After discovering this encouraging result, we decided to test our algorithm on third category namely, random flows on grid. We decided to test our algorithm on Nugent Instance set with sizes 15, 20, 25 and 30. Results of the experiment can be found on Table 4.26.

Table 4.27 : Nugent Instance Set Optimizing Number of Islands

Nugent15 Results of 10 Runs for each Configuration					
Number of Islands	Population per Island	Minimum RPC %	Mean RPC %	Trimmed Mean RPC %	Maximum RPC %
Single Population	600	8.35	10.59	10.65	12.35
2 - Islands	300	0.17	1.98	2.00	3.65
4 - Islands	150	1.57	2.61	2.52	4.35
8 - Islands	75	0.00	1.44	1.46	2.78
10 - Islands	60	0.00	1.29	1.26	2.78
12 - Islands	50	0.17	1.62	1.65	2.78
15 - Islands	40	0.00	1.46	1.48	2.78
20 - Islands	30	0.52	1.57	1.54	2.78
30 - Islands	20	0.17	1.58	1.57	3.13
40 - Islands	15	0.00	1.46	1.30	4.17
50 - Islands	12	0.00	1.74	1.65	4.17
60 - Islands	10	0.17	1.67	1.67	3.13
75 - Islands	8	0.87	1.88	1.85	3.13
Nugent20 Results of 10 Runs for each Configuration					
Single Population	600	11.67	13.50	13.55	14.94
2 - Islands	300	1.17	2.48	2.48	3.81
4 - Islands	150	1.17	2.56	2.50	4.44
8 - Islands	75	1.17	2.41	2.37	3.89
10 - Islands	60	1.17	2.96	3.05	4.05
12 - Islands	50	1.17	2.61	2.59	4.20
15 - Islands	40	1.17	3.00	3.00	4.83
20 - Islands	30	0.70	2.12	2.06	4.05

30 - Islands	20	1.09	2.40	2.35	4.05
40 - Islands	15	1.71	2.72	2.65	4.28
50 - Islands	12	1.32	2.49	2.48	3.74
60 - Islands	10	1.63	2.93	2.79	5.37
75 - Islands	8	0.39	3.10	3.19	5.06
Nugent25 Results of 10 Runs for each Configuration					
Single Population	600	15.28	16.43	16.39	17.90
2 - Islands	300	0.27	2.35	2.26	5.18
4 - Islands	150	0.27	2.44	2.45	4.49
8 - Islands	75	0.48	2.39	2.40	4.22
10 - Islands	60	0.16	2.31	2.23	5.08
12 - Islands	50	0.69	2.38	2.38	4.06
15 - Islands	40	0.32	2.25	1.98	6.36
20 - Islands	30	0.91	1.99	1.94	3.47
30 - Islands	20	0.80	2.57	2.62	3.95
40 - Islands	15	0.59	2.32	2.40	3.47
50 - Islands	12	0.37	2.28	2.28	4.22
60 - Islands	10	0.86	3.30	3.43	4.75
75 - Islands	8	2.14	3.57	3.45	5.98
Nugent30 Results of 10 Runs for each Configuration					
Single Population	600	15.87	17.90	18.03	18.88
2 - Islands	300	1.31	2.81	2.81	4.25
4 - Islands	150	1.73	3.20	3.16	4.96
8 - Islands	75	1.70	2.81	2.66	5.13
10 - Islands	60	1.83	3.12	3.10	4.57
12 - Islands	50	0.46	2.56	2.57	4.61
15 - Islands	40	1.63	3.08	3.09	4.44
20 - Islands	30	1.73	2.95	2.87	4.83
30 - Islands	20	1.11	2.40	2.34	4.21
40 - Islands	15	1.70	3.12	3.07	4.93
50 - Islands	12	0.98	2.57	2.45	5.10
60 - Islands	10	0.98	3.21	3.23	5.29
75 - Islands	8	0.98	3.56	3.57	6.07

We came across somewhat different results compared to previous experiments, possibly due to relatively smaller sizes of Nugent instances. While 50 island configuration still performs fairly, we cannot conclude on a common best configuration for all instances. Most crude inference we can make is any number of islands between 30 and 50 seems reasonable for this problem set. However, we notice

that even 2 Island configuration can find very good results compared to sequential version being at least 8, up to 15 times better.



CHAPTER 5

CONCLUSIONS AND FUTURE WORK

Quadratic Assignment Problem is one of the most studied and one of most difficult combinatorial optimization problems in the literature. Algorithms proposed for it ranges from branch and bound exact algorithm to metaheuristics like ant colony, simulated annealing, tabu search, genetic algorithms, break-out local search techniques. Metaheuristic methodologies are open to development and will continue to contribute to the literature, while a good metaheuristic balances out exploration of search space and exploitation of optimum points.

Parallelism plays an important role to achieve these goals in genetic algorithms, as well as other types of metaheuristics. A good parallelism, with communication between different runs of algorithm, creates a diversification within different runs and prevents them from stagnation or premature convergence, while different runs working simultaneously, allows searching of different portions of search space. This of course leads to better results. Parallelism also used to reduce completion time of algorithm. These factors are especially another huge advantage in combinatorial optimization where exact algorithms are insufficient to handle large instances.

Genetic algorithms are inherently parallel by their nature. There are two good strategies proposed in literature: island and cellular Models. In this thesis, we optimize a standard sequential genetic algorithm using island and master - slave models. Our results were encouraging. We were able to improve algorithms' performance by 2 to 15 times in terms of solution quality and optimized other parameters to find some reasonable island model configurations for different sets of instances with different characteristics.

The results of the experiments show that parallelization of a sequential GA using island model influences algorithm's performance. As a future work, more parameters and more parameters values can be considered in the experiments and methodologies like Design of Experiments or racing can be used to decide on the values.

The proposed algorithm also gains some speedup, although this speedup is sublinear. As a future work, the communication model can be improved using asynchronous communication instead synchronous model. This may better utilize the computing

power, since the islands will not have to block on waiting for their neighbors to send immigrants and continue the evolution process.

Different crossover and mutation operators, along with their parameters, of the serial GA can be experimented to select the best configuration to achieve better results.

In addition, cellular models can be used for parallelization of genetic algorithms. Unlike island models which are suitable with MIMD CPU units of today, cellular models are more compatible with SIMD GPU units. Another approach could be using a hybrid model.



REFERENCES

- A.Misevicius. (2003). A Modified Simulated Annealing Algorithm for the Quadratic Assignment Problem. *INFORMATICA*, 497–514.
- A.Misevicius. (2004). An improved hybrid genetic algorithm: new results for the quadratic assignment problem. *Knowledge-Based Systems*, 65–73.
- A.Misevicius. (2005). A Tabu Search Algorithm for the Quadratic Assignment Problem. *Computational Optimization and Applications*, 95–111.
- Ahuja, R. K., Orlin, J. B., & Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assignment. *Computers & Operations Research*, 917- 934.
- Aksan, Y., Dokeroglu, T., & Cosar, A. (2017). A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering*, 105–115.
- Alba, E. (2002). Parallel evolutionary algorithms can achieve super-linear performance. *Information processing letters*, 7-13.
- Alba, E., & Dorronsoro, B. (2008). A Hybrid Cellular Genetic Algorithm for the Capacitated Vehicle Routing Problem. *Engineering Evolutionary Intelligent Systems*, 379-422.
- Alba, E., Giacobini, M., Tomassini, M., & Romero, S. (2002). Comparing Synchronous and Asynchronous Cellular Genetic Algorithms. *International Conference on Parallel Problem Solving from Nature* (pp. 601-610). Berlin: Springer.
- Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *American Federation of Information Processing Societies (Spring)* (pp. 483-485). Atlantic City, New Jersey: ACM.
- Anstreicher, K. M., Brixius, N. W., Goux, J.-P., & Linderoth, J. (2002). Solving large quadratic assignment problems on computational grids. *Mathematical Programming* 91.3, 563-588.
- Armour, G., & Buffa, E. (1963). A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 294–304.
- Azab, A. (2015). Quadratic assignment problem mathematical modelling for process planning. *International Journal of Computer Integrated Manufacturing* , 561-580.
- Bachelet, V., Preux, P., & Talbi, E. G. (1996). Parallel hybrid meta-heuristics: application to the quadratic assignment problem. *Proceedings of the Parallel Optimization Colloquium*, (pp. 233-242).
- Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Second*

- International Conference on Genetic Algorithms on Genetic algorithms and their application (pp. 14-21). Cambridge ,MA: L. Erlbaum Associates Inc.
- Barney, B. (2009, January 29). Introduction to Parallel Computing: Part 1. Retrieved from <http://www.drdoobbs.com>: <http://www.drdoobbs.com/introduction-to-parallel-computing-part/212903586>
- Bazaraa, M. S., & Elshafei, A. N. (1979). An exact branch-and-bound procedure for the quadratic-assignment problem. *Naval Research Logistics*, 109-121.
- Bazaraa, M. S., & Sherali, H. D. (1982, November). On the Use of Exact and Heuristic Cutting Plane Methods for the Quadratic Assignment Problem. *Journal of the Operational Research Society*, 33(11), 991–1003.
- Bazaraa, M., & Kirca, O. (1983). A branch-and-bound-based heuristic for solving the quadratic assignment problem. *Naval Research Logistics* , 287-304.
- Benlic, U., & Hao, J. (2013). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 4800- 4815.
- Benlic, U., & Hao, J. K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 584–595.
- Bethke, A. D. (1976). Comparison of genetic algorithms and gradient-based optimizers on parallel processors: efficiency of use of processing capacity. University of Michigan, College of Literature, Science, and the Arts, Computer and Communication Sciences Department.
- Bölte, A., & Thonemann, U. (1996). Optimizing simulated annealing schedules with genetic programming. *European J. of Operational Research*, 402–416.
- Bougleux, S., Bruna, L., Carletti, V., Foggia, P., Gaüzère, B., & Vento, M. (2017). Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 38–46.
- Brixius, N., & Anstreichner, K. (2001). The Steinberg wiring problem. Iowa: The University of Iowa.
- Brungger, A., Marzetta, A., Clausen, J., & Perregaard, M. (1997). Joining forces in solving large-scale quadratic assignment problems in parallel. *Proceedings 11th International Parallel Processing Symposium* (pp. 418-427). Geneva: IEEE.
- Burkard, R. E., & Derigs, U. (1980). Quadratic Assignment Problems. In B. H.E., & u. Derigs, *Lecture Notes in Economics and Mathematical Systems 184 Assignment and Matching Problems: Solution Methods with FORTRAN-Programs* (pp. 99-146). Springer-Verlag Berlin.
- Burkard, R. E., & Offermann, J. (1977). Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme. *Zeitschrift für Operations Research*,

B121–B132.

- Burkard, R., & Rendl, F. (1984). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 169-174.
- Burkard, R., Karisch, S., & Rendl, F. (1997). QAPLIB – A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10(4), 391–403.
- C.Bousoño-Calzón, & Manning, M. R. (1995). The Hopfield neural network applied to the Quadratic Assignment Problem. *Neural Computing & Applications*, 64–72.
- Cantú-Paz, E. (1995). A summary of research on parallel genetic algorithms. *Illinois Genetic Algorithms Laboratory*, University of Illinois.
- Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 141-171.
- Çela, E. (1998). *The Quadratic Assignment Problem Theory and Algorithms*. Dordrecht: Kluwer Academic Publishers.
- Chakrapani, J., & Skorin-Kapov, J. (1993). Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 327-341.
- Christofides, N., & Benavent, E. (1989, October). An Exact Algorithm for the Quadratic Assignment Problem on a Tree. *Operations Research*, 37(5), 760-768.
- Connolly, D. (1990). An improved annealing scheme for the QAP. *European J. of Operational Research*, 93–100.
- Cormen, T. (2013). *Hard? Problems*. MIT Press.
- Cung, V. D., Mautor, T., Michelon, P., & Tavares, A. (1997). A scatter search based approach for the quadratic assignment problem. *IEEE International Conference on Evolutionary Computation*, (pp. 165-169).
- Davis, L. (1987). *Genetic Algorithms and Simulated Annealing*.
- Demirel, N., & Toksari, M. (2006). Optimization of the quadratic assignment problem using an ant colony algorithm. *Applied Mathematics and Computation*, 427–435.
- Dickey, J., & Hopkins, J. W. (1972). Campus building arrangement using TOPAZ. *Transportation Research*, 59-68.
- Dorigo, M., Birattari, M., & Stützle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 28-39.
- Dyer, D. W. (2013, 1 31). *Watchmaker Framework for Evolutionary Computation*. Retrieved from Github: <https://github.com/dwdyer/watchmaker>
- Edwards, C. (1980). A branch and bound algorithm for the Koopmans-Beckmann quadratic assignment problem. *Combinatorial Optimization II*, 35-52.

- Eiben, A., & Smith, J. (2003). *Introduction to Evolutionary Computing*. Berlin: SpringerVerlag .
- Elshafei, A. N. (1977). Hospital Layout as a Quadratic Assignment Problem. *Journal of the Operational Research Society*, 167–179.
- Eshelman, L. J. (1997). Genetic algorithms. In T. Back, D. Fogel, & Z. Michalewicz, *Handbook of Evolutionary Computation* (pp. 44-54). Bristol: IOP Publishing Ltd.
- Ettinger, T. (2017, 05 15). <https://github.com/SquidPony/SquidLib>. Retrieved from Github: <https://github.com/SquidPony/SquidLib>
- Fan, Y., Jiang, T., & Evans, D. (2002). Volumetric segmentation of brain images using parallel genetic algorithms. *IEEE Transactions on Medical Imaging*, 904 - 909.
- Feo, T., & Resende, M. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 109-133.
- Fleurent, C., & Ferland, J. A. (1993). Genetic hybrids for the quadratic assignment problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 173-187.
- Flynn, M. J. (1972, September). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9), 948-960. doi:10.1109/TC.1972.5009071
- Folino, G., Pizzuti, C., & Spezzano, G. (1998). Solving the satisfiability problem by a parallel cellular genetic algorithm. *Proceedings. 24th EUROMICRO Conference (Cat. No.98EX204)* (pp. 715-722). Vasteras,SWE: IEEE.
- Francis, R. L., Jr., F. M., & White, J. A. (1974). *Facility Layout and Location: An Analytical Approach*. Englewood Cliffs , NJ: Prentice Hall.
- Gambardella, L. M., Taillard, É., & Dorigo, M. (1999). Ant Colonies for the Quadratic Assignment Problem. *The Journal of the Operational Research Society*, 167-176.
- Gavett, J. W., & Plyter, N. V. (1966). The Optimal Assignment of Facilities to Locations by Branch and Bound. *Operations Research*, 210 - 232.
- George, A., & Pothen, A. (1997). An Analysis of Spectral Envelope Reduction via Quadratic Assignment Problems. *SIAM Journal on Matrix Analysis and Applications*, 706–732.
- Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing* , 190-206.
- Glover, F. (1990). Tabu Search—Part II. *ORSA Journal on Computing*, 4-32.
- Glover, F., Laguna, M., & Marti, R. (2003). Scatter Search. *Advances in evolutionary computing*, 519-537.
- Goldberg, D. (1988). Genetic Algorithms in Search, Optimization and Machine Learning. *Machine Learning* 3, 95-99.

- Goldberg, D., & Lingle, R. (1985). Alleles, Loci and the Traveling Salesman Problem. 1st International Conference on Genetic Algorithms (pp. 154-159). Cambridge,MA: L. Erlbaum Associates Inc.
- Gordon, V. S., & Whitely, D. (1993). Serial and Parallel Genetic Algorithms as Function Optimizers. ICGA, (pp. 177-183).
- Gorges - Schleuter, M. (1989). ASPARAGOS A Parallel Genetic Algorithm and Population Genetics. Workshop on Parallel Processing :Logic Organization and Technology (pp. 413-424). Wildbad Kreuth, Germany: Springer-Verlag.
- Grefenstette, J. J. (1981). Parallel Adaptive Algorithms for Function Optimization. Computer Science Department, Vanderbilt University.
- Grosso, P. (1985). "Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model.
- Hafiza, F., & Abdennourb, A. (2016). Particle Swarm Algorithm variants for the Quadratic Assignment Problems - A probabilistic learning approach. Expert Systems With Applications, 413–431.
- Hahn, P., Hightower, W., Johnson, T., Guignard-Spielberg, M., & Roucairol, C. (2001). Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. Yugoslav Journal of Operations Research, 41-60.
- Hansen, P., & Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. European Journal of Operational Research, 449-467.
- Heragu, S. S., & Alfa, A. S. (1992). Experimental analysis of simulated annealing based algorithms for the layout problem. EJORDT, 190-202.
- Holland, J. H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Ann Arbor, MI: University of Michigan Press.
- Hubert, L., & Schultz, J. (1976). Quadratic Assignment as a General Data Analysis Strategy . British Journal of Mathematical and Statistical Psychology, 190–241.
- Ibounig, T. (2016, 04 29). PRNG-Performance. Retrieved from Github: <https://github.com/tobijdc/PRNG-Performance>
- James, T., Rego, C., & Glover, F. (2009). cooperative parallel tabu search algorithm for the quadratic assignment problem. European Journal of Operational Research, 810–826.
- James, T., Rego, C., & Glover, F. (2009). Multistart Tabu Search and Diversification Strategies for the Quadratic Assignment Problem. IEEE Transactions on Systems ,Man and Cybernetics , PartA :Systems and Humans, 579–596.
- Kennedy, J. (2010). Particle Swarm Optimization. In C. Sammut, & G. Webb,

- Encyclopedia of Machine Learning (pp. 760-766). New York: Springer US. doi:10.1007/978-0-387-30164-8_630
- Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science, New Series*, 671-680.
- Koopmans, T., & Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, 53-76.
- Krarpup, J., & Pruzan, P. M. (1978). Computer-aided layout design. *Mathematical Programming Studies*, 75-94.
- Kurdi, M. (2016). An effective new island model genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 132-142.
- Laporte, G., & Mercure., H. (1988). Balancing hydraulic turbine runners: A quadratic assignment problem. *European Journal of Operational Research*, 378-381.
- Laursen, P. S. (1993). Simple approaches to parallel Branch and Bound. *Parallel Computing*, 143-152.
- Levine, D. (1994). A parallel genetic algorithm for the set partitioning problem.
- Li, Y., Pardalos, P. M., & Resende, M. G. (1994). A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 237-261.
- Liu, Y., & Wang, S. (2015). A scalable parallel genetic algorithm for the Generalized Assignment Problem. *Parallel Computing*, 98-119.
- Loiola, E. M., de Abreu, N. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007, January 16). A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2), 657–690.
- Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated Local Search. In F. Glover, & G. Kochenberger, *Handbook of Metaheuristics* (pp. 320-353). New York: Springer US. doi:10.1007/b101874
- Luque, G., Alba, E., & Dorronsoro, B. (2009). An asynchronous parallel implementation of a cellular genetic algorithm for combinatorial optimization. *GECCO '09 Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (pp. 1395-1402). Montreal,Canada: ACM New York.
- Maniezzo, V., & Colomi, A. (1994). The Ant System Applied to the Quadratic Assignment Problem. *IEEE Transactions on Knowledge and Data Engineering*, 769-778 .
- Mühlbein, H. (1989). Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization. *Workshop on Parallel Processing: Logic, Organization, and Technology* (pp. 398-406). Wildbad Kreuth, Germany:

Springer-Verlag.

- Mühlenbein, H., Schomisch, M., & Born, J. (1991). The parallel genetic algorithm as function optimizer. *Parallel Computing*, 619-632.
- Nissen, V. (1994). Solving the Quadratic Assignment Problem with Clues from Nature. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 66-72.
- Nyström, M. (1999). Solving Certain Large Instances of the Quadratic Assignment Problem: Steinberg's Examples. CA,US: California Institute of Technology.
- Ochi, L., Vianna, D., Drummond, L., & Victor, A. (1998). A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. *Future Generation Computer Systems*, 285-292.
- Padberg, M., & Rinaldi, G. (1991). A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM*, 60–100.
- Pardalos, P. M., & Crouse, J. V. (1989). A parallel algorithm for the quadratic assignment problem. *ACM/IEEE Conference on Supercomputing 89* (pp. 351-360). IEEE.
- Pardalos, P., Rendl, F., & Wolkowicz, H. (1994). The Quadratic Assignment Problem A Survey and recent developments. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1-42.
- Perreira, C., & Lapa, C. (2003). Coarse-grained parallel genetic algorithm applied to a nuclear reactor core design optimization problem. *Annals of Nuclear Energy*, 555-565.
- Phillips, A. T., & Rosen, J. B. (1994). A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization*, 229–241.
- R.Sagban, Ku-Mahamud, K., & Abu Bakar, M. (2017). Reactive max-min ant system with recursive local search and its application to TSP and QAP. *Intelligent Automation & Soft Computing*, 127-134.
- Robertson, G. (1987). Parallel implementation of genetic algorithms in a classifier system. *Genetic algorithms and their applications : proceedings of the second International Conference on Genetic Algorithms* (pp. 140-147). Cambridge, MA: Hillsdale, N.J. : L. Erlbaum Associates.
- Roucairol, C. (1987). A parallel branch and bound algorithm for the quadratic assignment problem. *Discrete Applied Mathematics*, 211-225.
- Rucinski, M., Izzo, D., & Biscani, F. (2010). On the impact of the migration topology on the Island Model. *Parallel Computing*, 555-571.
- Sahni, S., & Gonzales, T. (1976, July). P-Complete Approximation Problems. *Journal of the ACM*, 23(3), 555-565.

- Schwefel, H.-P. (1997). Advantages (and disadvantages) of evolutionary computation over other approaches. In T. Back, D. Fogel, & Z. Michalewicz, *Handbook of Evolutionary Computation* (pp. 14-15). Bristol: IOP Publishing Ltd. .
- Shapiro, B., & Navetta, J. (1994). A massively parallel genetic algorithm for RNA secondary structure prediction. *The Journal of Supercomputing*, 195-207.
- Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 33-45.
- Steinberg, L. (1961). The backboard wiring problem: a placement algorithm. *SIAM Review*, 37-50.
- Steiner, T., Jin, Y., & Sendhoff, B. (2008). A cellular model for the evolutionary development of lightweight material with an inner structure. *GECCO '08 Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 851-858). Atlanta, GA, USA: ACM New York.
- Stützle, T. (1999). Iterated local search for the quadratic assignment problem. *FG INTELLEKTIK, FB INFORMATIK*.
- Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 1519–1539.
- Stützle, T., & Hoos, H. (2000). MAX—MIN ant system. *Future Generation Computer Systems*, 889–914.
- Sudholt, D. (2015). *Parallel Evolutionary Algorithms*. In J. Kacprzyk, & W. Pedrycz, *Springer Handbook of Computational Intelligence* (pp. 929-957). New York City: Springer Publishing Company.
- Taillard, É. (1995). Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 87-105.
- Taillard, R. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 443-455.
- Talbi, E., Roux, O., Fonlupt, C., & D.Robillard. (2001). Parallel Ant Colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 441–449.
- Tanese, R. (1987). Parallel genetic algorithms for a hypercube. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (pp. 177-183). Hillside , NJ: Lawrence Erlbaum.
- Tanese, R. (1989). Distributed genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 434-439). San Mateo, CA: Morgan Kauffmann.
- Tang, J., Lim, M., Ong, Y., & Er, M. (2004). Study of migration topology in island model parallel hybrid-GA for large scale quadratic assignment problems. *Control*,

- Automation, Robotics and Vision Conference (pp. 2286-2291). Kunming, China: IEEE.
- Tate, D. M., & Smith, A. E. (1995). A Genetic Approach To The Quadratic Assignment Problem. *Computers Ops. Research.*, 73-83.
- Tosun, U., Dokeroglu, T., & Cosar, A. (2013). A robust Island Parallel Genetic Algorithm for the Quadratic Assignment Problem. *International Journal of Production Research*, 4117–4133.
- Tseng, L., & Liang, S. (2006). A Hybrid Metaheuristic for the Quadratic Assignment Problem. *Computational Optimization and Applications*, 85–113.
- U.Tosun. (2015). On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 267–278.
- Whitely, D., & Starkweather, T. (1990). GENITOR II: a distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 189-214.
- Woeginger, G. J. (2003). Exact Algorithms for NP-Hard Problems: A Survey. In M. Jünger, G. Reinelt, & G. Rinaldi, *Combinatorial optimization - Eureka, you shrink!* (pp. 185-207). New York, NY: Springer-Verlag New York, Inc.
- Wolpert, D., & Macready, W. (1997, April). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67-82. doi:10.1109/4235.585893