VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rámec pro plánování problémy

Framework for Scheduling Problems

2015 Magdalena Metlická

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Diploma Thesis Assignment

Student: **Bc. Magdalena Metlická**

Study Programme: N2647 Information and Communication Technology

Study Branch: 2612T025 Computer Science and Technology

Title: Framework for Scheduling Problems
Rámec pro plánování problémy

The thesis language: English

Description:

Scheduling problems encompass some of the most practical applications in manufacturing systems. A number of scheduling problems exist, including flowshop, jobshop, just-in-time manufacturing etc.
This thesis will require the development of a scheduling framework for selected heuristics. The framework will be encompass three unique aspects; constructive heuristics, improvement heuristics and scheduling problems.
The constructive heuristics will be:
1. NEH algorithm (and varients).
2. Local Search (2-OPT).
The improvement heuristics will be:
1. Discrete Artificial Bee Colony (DABC).
2. Chaos driven DABC.
The scheduling problems to be solved would be:
1. Permutative flowshop scheduling.
2. Flowshop scheduling with blocking constraint.
3. Flowshop scheduling with no-wait criterion.
All the constructive and improvement algorithms will be coded for both the CPU and GPU systems. The GPU framework should be developed on the CUDA platform optimized for the Kepler architecture.
The output of the thesis will contain the following:
1. CUDA accelerated constructive and improvement heuristics.
2. Extensive experimentation to show the performance of constructive and improvement heuristics for the different scheduling problems.
3. Analysis on the effect of chaos maps on DABC algorithm.
4. Complex network analysis of Artificial Bee Colony.

References:

[1] Nawaz, M., Enscore Jr., E and Ham, I. 1983. "A heuristic algorithm for the m- machine, n-job flow-shop sequencing problem," Omega, vol. 11, no. 1, pp. 91–95.
[2] Ruiz, R and Maroto, C. 2005. "A comprehensive review and evaluation of permutation flowshop heuristics." European Journal of Operational Research, vol. 165, pp. 479 – 494.
[3] Lin, S and Kernighan, B. 1973. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". Operations Research, vol. 21, no. 2, pp. 498–516
[4] Pinedo, M., 1995. Scheduling: theory, algorithms and systems. Prentice Hall, Inc., New Jersey.
[5] Baker, M and Triestsch, T., 2009. Principles of Sequencing and Scheduling. Wiley, USA.

[6] Brucker, P., 2007. Scheduling Algorithms. Springer, Germany.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. MSc. Donald David Davendra, Ph.D.**

Date of issue: 01.09.2014

Date of submission: 07.05.2015

doc. Dr. Ing. Eduard Sojka
Head of Department

prof. RNDr. Václav Snášel, CSc.
Dean of Faculty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 7. Května 2015 .....

I would like to express my gratitude to my supervisor associate professor Donald David Davendra for his useful comments, guidance and support. I would like to thank to my family for their support, namely to my mother Marie Metlická, and my grandmother Marie Hutterová, for their encouragement, and to my father Jaroslav Metlický, for his help with the language.

Abstrakt

Rozvrhovací problémy jsou důležitou podtřídou úloh kombinatorické optimalizace s řadou aplikací ve výrobě a logistice. Většina těchto problémů je NP-úplných (rozhodovací forma) a NP-těžkých (optimalizační forma), proto se výzkum zaměřuje na návrh efektivních heuristických algoritmů. Dvě hlavní kategorie těchto algoritmů jsou deterministické algoritmy a evoluční metaheuristiky. Deterministické algoritmy zahrnují techniky lokálního prohledávání, například algoritmus k-opt, jejichž cílem je zlepšení existujícího přípustného řešení problému, dále pak konstruktivní heuristiky, jejichž příkladem je algoritmus NEH, které hledané řešení vytvářejí inkrementálně, bez potřeby znalosti vstupního bodu v prohledávaném prostoru řešení. Evoluční metaheuristiky mají za sebou historii úspěšného vývoje v posledních desetiletích, zejména díky jejich efektivitě a flexibilitě. Jejich inspirací jsou poznatky převzaté z biologie, teorie evoluce a inteligence hejna. Mezi nejpopulárnějšími z těchto algoritmů jsou, mimo jiné, genetické algoritmy, diferenciální evoluce, rojení částic (Particle Swarm Optimisation).

Ačkoli tyto heuristiky nalézají ve většině případů řešení blížící se globálnímu optimu v přípustném výpočetním čase, pro řadu aplikací mohou být stále ještě nepřijatelně pomalé. Velké úsilí bylo věnováno zrychlení těchto algoritmů. Protože se vývoj hardware díky dosažení technologických limitů, vzhledem ke zvyšující se spotřebě energie a tepelnému vyzařování, obrací od zvyšování frekvence jednojádrového procesoru k vícejádrovým procesorům a paralelnímu zpracování, je tato snaha většinou orientovaná na paralelizaci existujících algoritmů, aby bylo umožněno využití výpočetní síly vícejádrových platforem (multi-core a many-core). Prvním cílem této práce je tudíž akcelerace dvou deterministických algoritmů, NEH a 2-opt, přičemž bylo dosaženo zajímavých výsledků.

Jiný přístup byl zvolen ve druhé části, s hlavní myšlenkou prozkoumání vlivu náhodnosti na výkon evolučního algoritmu. Za tímto účelem byl zvolen relativně nový a slibný algoritmus Discrete Artificial Bee Colony. Generátor pseudonáhodných čísel byl nahrazen několika různými chaotickými mapami, z nichž některé znatelně zlepšily výsledky algoritmu.

V příspěvcích [27] a [26] bylo ukázáno, že evoluční algoritmy založené na populaci často formují komplexní sítě, vzato z pohledu výměny informací mezi jednotlivými řešeními v populaci během jejího vývoje. Závěrečná část práce aplikuje toto pozorování vložením samo přizpůsobivého mechanismu založeném na analýze komplexní sítě do algoritmu ABC, který je evolučním algoritmem pro spojitou optimalizaci a zároveň základem dříve zmíněného DABC algoritmu. Efektivita několika verzí algoritmu založeném na této myšlence je dokázána na standardní sadě testovacích funkcí pro spojitou optimalizaci. Možnost rozšíření této modifikace na kombinatorické optimalizační problémy je diskutována v závěru práce.

Klíčová slova: Plánování, Rozvrhování, Kombinatorická optimalizace, Spojitá optimalizace, Heuristiky, Evoluční algoritmy, Deterministické heuristiky, Flowshop, Flowshop s no-wait omezením, Lot-streaming flowshop, Quadratic Assignment Problem, Capacitated Vehicle Routing Problem, Artificial Bee Colony, Discrete Artificial Bee Colony, NEH, 2-opt, Chaos, Chaotické mapy, Komplexní sítě, CUDA

Abstract

Scheduling problems form an important subclass of combinatorial optimisation problems with many applications in manufacturing and logistics. Predominately these problems are NP-complete (decision based) and NP-hard (optimisation based), hence the main course of research in solving them concentrates on the design of efficient heuristic algorithms. Two main categories of these algorithms exist: deterministic algorithms and evolutionary metaheuristics. The deterministic algorithms comprise local improvement techniques, such as k-opt algorithm, which try to improve existing feasible solution, and constructive heuristics, such as NEH, which build a solution starting from scratch, adding one job at a time. Evolutionary metaheuristics have prospered in the past decades, owing to their efficiency and flexibility. Drawing inspiration from the theory of natural evolution or swarm behavioural patterns, the most popular of these algorithms in practice include for instance Genetic Algorithms, Differential Evolution, Particle Swarm Optimisation, amongst others.

However, even though these heuristics provide in most cases close to optimal solution at reasonable execution time, this time is still impractically long for many applications. Therefore much effort has been dedicated to accelerating these algorithms. Since the development of hardware turns away from increasing the clock speed towards the parallel processing units, owing to reaching the limits of technology due to the increased power consumption and heat dissipation, this effort goes into parallelisation of the existing algorithms, to enable exploitation of the computing power of multi-core or many-core platforms. This is the goal of the first part of the thesis, accelerating two of the deterministic algorithms, NEH and 2-opt, with interesting results.

Another approach has been taken in the second part, with the core premise of exploring the influence of stochasticity on the performance of an evolutionary algorithm, selecting the relatively recent and promising Discrete Artificial Bee Colony algorithm. The pseudo-random number generator has been replaced with the different types of dissipative chaos maps, with some of them improving the algorithm significantly.

It has been shown in [27] and [26], that the population based evolutionary algorithms often form complex networks, taken from the point of view of the information exchange between individual solutions during the course of population development. The final part of this thesis puts this observation into practice by embedding the complex network

analysis based self-adaptive mechanism into the ABC algorithm, a continuous optimisation problems solving evolutionary algorithm, which is however the basis for the afore mentioned DABC algorithm, and proving the effectiveness for some of the developed versions, currently on the standard continuous optimisation test functions, with the possibility to extend this modification to the combinatorial optimisations problems in the future being discussed in the conclusion.

Keywords: Scheduling, Combinatorial Optimisation, Continuous Optimisation, Heuristics, Evolutionary Algorithms, Deterministic Heuristics, Flowshop, Flowshop with No-Wait, Lot-Streaming Flowshop, Quadratic Assignment Problem, Capacitated Vehicle Routing Problem, Artificial Bee Colony, Discrete Artificial Bee Colony, NEH, 2-opt, Chaos, Chaos maps, Complex Networks, CUDA

List of Abbreviations and Symbols

ABC	–	Artificial Bee Colony
AC	–	Arnold's Cat map
ACO	–	Ant Colony Optimisation
AIA	–	Artificial Immune Algorithm
API	–	Application Programming Interface
B	–	Burgers map
CDABC	–	Chaos driven Discrete Artificial Bee Colony
$CDABC_B$	–	Chaos driven Discrete Artificial Bee Colony, Burgers map
$CDABC_{DL}$	–	Chaos driven Discrete Artificial Bee Colony, Delayed Logistic
$CDABC_L$	–	Chaos driven Discrete Artificial Bee Colony, Lozi map
$CDABC_T$	–	Chaos driven Discrete Artificial Bee Colony, Tinkerbell map
EDE_C	–	Chaos driven Enhanced Differential Evolution
CN	–	Complex Networks
CPRNG	–	Chaos Pseudo-Random Number Generator
CPU	–	Central Processing Unit
CUDA	–	Compute Unified Device Architecture
CVRP	–	Capacitated Vehicle Routing Problem
DABC	–	Discrete Artificial Bee Colony
$DABC_{MT}$	–	Discrete Artificial Bee Colony, Mersenne Twister
DE	–	Differential Evolution
Dis	–	Dissipative map
DL	–	Delayed Logistic
DSOMA	–	Discrete Self-Organizing Migrating Algorithm
EA	–	Evolutionary Algorithm
EDE	–	Enhanced Differential Evolution
ETS	–	Extended Taillard Data sets
FSS	–	Flow Shop Scheduling
FSSLS	–	Flow Shop Scheduling with Lot Streaming
FSSNW	–	Flow Shop with No-Wait
FSSZIS	–	Flow Shop with Zero Intermediate Storage
GA	–	Genetic Algorithm
GPGPU	–	General Purpose GPU computing
GPU	–	Graphic Processing Unit

Hen	–	Henon map
HPC	–	High-performance computing
I	–	Ikeda map
L	–	Lozi map
MT	–	Mersenne Twister
NEH	–	Nawaz, Enscore, Ham heuristic
NP	–	Nondeterministic Polynomial Time
OTS	–	Original Taillard Data sets
PL1D	–	Chaotic Piecewise-Linear One Dimensional
PRD	–	Percentage Relative Difference
PRNG	–	Pseudo-Random Number Generator
PSO	–	Particle Swarm Optimisation
QAP	–	Quadratic Assignment Problem
S	–	Sinai map
SM	–	Streaming Multiprocessor
SOMA	–	Self-Organizing Migrating Algorithm
STD	–	Standard Deviation
T	–	Tinkerbell map
TFT	–	Total Flow Time
TSP	–	Travelling Salesman Problem

Contents

List of Tables

List of Figures

1 Introduction

Scheduling is an everyday term, which infers on the arrangement of some process, task or resources. These are generally referred to as schedules, which can be defined as a tangible plan or document. A schedule is this sense can range from a bus timetable to a flight schedule.

A structured schedule is an integral part of any engineering or manufacturing process. The main emphasis therefore from the engineering point of view is knowing the type and the amount of each resource so that we can determine when the tasks can feasibly be accomplished. The resource in this way define the schedule boundaries, therefore leading to a bounded solution space. Once the resources have been defined, it becomes imperative to describe the tasks in terms of resource requirements, duration and start time.

Scheduling theory deals with mathematical models that relate to the process of scheduling. This is considered a quantitative approach, where the problem structure is converted into a mathematical form. The main emphasis is then to translate the description of resources and tasks to an explicit objective function [5].

The objective function of any scheduling system relies on three facets; *turnaround*, which measures the time required to complete the task, *timeliness*, which measures the conformance of a particular task's completion to a given deadline and *throughput*, which measures the amount of work completed during a fixed period of time.

A particular *solution* to a scheduling problem has to take into consideration two issues [5]:

1. resource allocation for a particular task
2. allocation of particular task

If the resources are available at the beginning of the task, the schedule is considered *deterministic*. If the resources become available over time during the task, the system is then considered *stochastic*.

A number of different models exist which solve scheduling problems. These can be divided into two categories: *deterministic* and *heuristic* models.

1.1 Deterministic Algorithms

Deterministic models have been around since the mid 1940's, when the advent of World War 2, led to the establishment of the operations research discipline, in order to conserve and carefully utilise dwindling resources. Some of the most widely used models can be classified as the following:

1. Linear Programming (LP) model [100]
2. Simplex Model [19]
3. NEH Algorithm [76]

4. Lin–Kernighan algorithm (k-opt) [104]

The LP model is the most widely used general scale model. Its corresponding discrete counterpart is the Integer Programming (IP) model or the Integer Linear Programming (ILP) model. The LP model is constructed using three basic elements:

1. Decision variables: which represents (unknown) decisions to be made, which is in contrast to the *problem* data, which is precise.

2. Objective: which has to be *linear* on the decision variables, implying that it is the sum of *constraints* times the *decision variables*.

3. Constraints: which limit *feasible* decisions.

Using the above three elements, the final model of a problem can be constructed. This model can be solved using a number of different approaches, such the algebraic, graphical or simplex method.

The simplex method is one the most powerful approaches to solve a LP model. Using the standard form of LP model, a simplex tableaux can be constructed. By generating a basic feasible solution, repeated iterations can be conducted on the tableaux in order to find the optimal value of the entering variables. For full description of the simplex algorithm, the reader is referred to [19]. Many different approaches of simplex exists, including the revised simplex algorithm.

Whereas, simplex uses a basic feasible solution to start its iterations, the NEH (Nawaz, Ensore and Ham) algorithm aims to construct a near optimal solution. The NEH model relies on two preconditions; that the resources are available at the start and the processors are preemptive. The principle of NEH is quite simple:

1. Compute the completion time of all tasks using the resources.

2. Sort the tasks in ascending order based on the completion time.

3. Take the first two tasks and sort them to satisfy the objective function.

4. Likewise take each subsequent remaining task and sort it with the partial completed task list until all tasks are hence scheduled.

The NEH algorithm is widely considered the highest performing method in manufacturing scheduling and is an inherent component of almost all major scheduling systems [95].

The Lin–Kernighan algorithm on the other hand tries to exploit the neighbourhood of a schedule in order to improve it. The precondition of this algorithm is that the schedule must be complete. The steps can be given as:

1. Take the first task and swap it with the adjacent task.

2. Evaluate the new schedule.

3. If the objective function is improved, then retain the new schedule.

4. Iterate for all subsequent tasks in the schedule.

A number of variants of this algorithms exists, and are generally identified by their complexity. A 2-opt algorithm swaps two tasks in the schedule at one time, and is considered the canonical variant.

1.2 Heuristic Algorithms

Since the advent of computing resources, heuristics have come to dominate many engineering applications. Some of the first constructive heuristics to be successfully applied to scheduling problems were:

- Tabu Search [16]
- Simulated Annealing [54]

In the late 1990's however, meta-heuristics based on evolutionary paradigms started to appear. Algorithms based on naturally occurring phenomena, such as human genetics, population demographics, swarm behaviour and patterns started to evolve and have become the most dominant paradigms within the last decade. Some of the most common ones are:

- Genetic Algorithms [93]
- Differential Evolution [33]
- Particle Swarm Optimisation [112]
- Self-Organising Migrating Algorithm [17]
- Harmony Search Algorithm [86]
- Artificial Bee Algorithm [48]
- Firefly Algorithm [31]
- Bat Algorithm [124]

1.3 Scheduling framework

As scheduling covers such a broad and diverse field of applications, this thesis is concentrated on three main classes of manufacturing, routing and assignment problems.

Manufacturing scheduling is concerned with the manufacturing process, where jobs are generally scheduled to machines in order to reduce the total completion time (makespan) or to minimise delay (tardiness).

A general manufacturing environment is generally refereed to as a *shop* and the formulation of the process; the way in which the jobs are completed is referred to as its

complexity. Generally, all such environments can be contrived as strictly permutative and are hence therefore as least NP Complete [88].

The three manufacturing problems solved in this thesis are the permutative flowshop scheduling (FSS) problem, the flowshop with no wait (FSSNW) problem and the lot streaming flowshop (FSSLS) problem.

Routing problems have gained importance in fleet scheduling, airport routing and delivery scheduling amongst others. The capacitated vehicle routing problem (CVPR) [115] is where a fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost.

The quadratic assignment problem (QAP) generally deals with two counterparts; facilities and its locations. As there is a natural flow between these facilities, the objective is then to assign all facilities to different locations in such a way as to minimize the sum of the distances multiplied by the corresponding flows.

The main meta-heuristic used in this thesis is the Artificial Bee Colony (ABC) [47] and its discrete variant Discrete Artificial Bee Colony (DABC) [114]. This algorithm was selected as it is one of the most recent and promising algorithms, with a wide array of procedures, which in turn makes it quite versatile in solving scheduling problems. The main aim has been to modify and improve this heuristic.

To archive this aim, two separate experimentations were conducted, one on chaos based stochasticity and the other on complex network analysis. In regards to complex network analysis, the initial development was conducted on the canonical ABC algorithm. In order to test this algorithm and its self adaptive features, unimodal and multimodal real-domain problems were tested, and thereby included in this thesis.

The framework consists of three unique aspects.

GPU Acceleration : the first aspect of the thesis is the acceleration of the NEH and 2-Opt local search algorithms. As both these deterministic algorithms have proven to be time demanding, CUDA based GPU accelerated variants have been developed and tested on standard test sets.

Chaos stochasticity : Unique chaotic maps have been added to the DABC algorithm to gauge its effectiveness and analyse the influence of stochasticity.

Complex Network Analysis : A self-adaptive islands model based ABC algorithm is developed, where complex network properties, especially centralities are measured during evolutions. These measures are then used to develop self-adaptive mechanism to steer the population development.

The description of the experimentations is given in Table 1.

The thesis outline is as follows. Chapter 2 gives the theoretical background of different evolutionary algorithms, different chaos maps, description of complex network analysis, an overall introduction to high performance computing and CUDA and the test problem formulation.

Chapter 3 gives the detailed description of all the improvements and implementation of the algorithms. Section 3.1 outlines the CUDA accelerated NEH algorithm and section

Table 1: Outline

Paradigm	Heuristic	Problem
CUDA (GPU)	NEH 2-Opt	FSS[73] FSS
Chaos maps	Discrete Artificial Bee Colony	FSSNW[74] FSSLS[71] QAP[75] CVRP[75]
Complex Network	Artificial Bee Colony	Standard test functions[72]

3.2 describes the CUDA based 2-opt local search. The chaos based DABC algorithm is described in 3.3. The final component of complex network embedded ABC algorithm is given in section 3.4.

All experimentation and results is given in Chapter 4, with the CUDA accelerated NEH given in section 4.1 and 2-opt local search in section 4.2. The four experimentations of chaos induced DABC algorithm is given as follows: FSSLS in section 4.3, FSSNW in section 4.4, QAP in section 4.5 and CVRP in section 4.6. The results for the complex network analysis is given in section 4.7.

The analysis of the experimentations is given in Chapter 5 and follows the same trend as the experimentation section. CUDA accelerated NEH is analysed in section 5.1 and 2-opt local search in section 5.2. The experiments for the CDABC algorithm is analysed in section 5.3 for the FSSLS problem, section 5.4 for the FSSNW problem, section 5.5 for the QAP problem and section 5.6 for the CVRP problem. The analysis of complex network is given in section 5.7.

The thesis is finally concluded in Chapter 6.

2 Theory

2.1 Evolutionary Algorithms and Heuristics

In the following sections the overview of heuristics and evolutionary algorithms used in this research is presented. For the first part of the thesis, which is the acceleration of the selected heuristics used in scheduling problems, the NEH algorithm is described in Section 2.2, and the 2-opt algorithm in Section 2.3. The second part concentrates on the enhancement of an evolutionary algorithm applied to the scheduling problem by employing chaos pseudo-random number generator in place of standard pseudo-random number generators. The evolutionary algorithm chosen for this purpose was the efficient population based metaheuristic specifically developed for the solution of combinatorial optimisation problems, the DABC algorithm, hence its description is presented in Section 2.5. Finally for the last part, an evolutionary algorithm was enhanced with the complex network analysis based self-adaptive control. For this research, the original ABC algorithm for continuous optimisation, a basis for the later developed DABC algorithm, was used. Therefore, the ABC algorithm description is presented in Section 2.4. Because the DABC was created as a modification to the original continuous optimisation technique ABC and uses the same basic structure, the description of ABC algorithm precedes the DABC in the text.

2.2 NEH Algorithm

The NEH algorithm can be described as follows. Assume that $p_{i,j}$ can be considered as the processing time of job j on machine M_i, where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. The objective is to minimise the makespan, which can be represented as the length of a critical path in an acyclic network.

NEH can be described by the following steps:

Step 1: Arrange the jobs by decreasing sums of their total processing times $T_k = \sum_{i=1}^{m} p_{i,k}$.

Step 2: Take the first two jobs, find their order with the shorter makespan, and set $L = 3$.

Step 3: Assume that the current subsequence is $(j_1, j_2, \ldots, j_{L-1})$. Find the one with the shortest makespan in $(r, j_1, j_2, \ldots, j_{L-1}), (j_1, r, j_2, \ldots, j_{L-1}), \ldots,$ $(j_1, j_2, \ldots, j_{L-1}, r)$

Step 4: Set $L =: L + 1$. If $L = n + 1$, then stop; otherwise return to Step 3.

The general complexity of NEH can be given as $O\left(mn^2\right)$ [46].

2.3 2-opt Algorithm

The 2-opt algorithm is one of the most famous heuristics developed originally for solving the TSP problem. It was first proposed by Croes [15]. Along with 3-opt, generalized as k-

opt [62], these heuristics are based on exchange of up to k edges in a TSP tour (more information on application of k-opt local search techniques to TSP and CRP problems can be obtained from [99]). Together they are called exchange or local improvement heuristics. The exchange is considered to be a single move, from this point of view, such heuristics search the neighbourhood of the current solution, i.e. perform a local search and provide a locally optimal solution (k-optimal) to the problem [45].

The 2-opt procedure requires a starting feasible solution. It then proceeds by replacing the two non-adjacent edges, (v_i, v_{i+}) and (v_j, v_{j+}) by (v_i, v_j) and (v_{i+}, v_{j+}), and reversing one of the subpaths produced by dropping of edges, in order to maintain the consistent orientation of the tour. For example, the subpath $(v_i, v_{i+}, \ldots, v_j, v_{j+})$ is replaced by $(v_i, v_j, \ldots, v_{i+}, v_{j+})$. The solution cost change produced in this way can be expressed as $\Delta_{ij} = c(v_i, v_j) + c(v_{i+}, v_{j+}) - c(v_i, v_{i+}) - c(v_j, v_{j+})$. If $\Delta_{ij} < 0$, the solution produced by the move improves upon its predecessor. The procedure iterates until no move where $\Delta_{ij} < 0$ (no improving move) can be found [38].

In the scheduling problem, the 2-opt move constitutes of exchanging two jobs of the schedule.

The 2-opt local search was described by Kim, Shim and Zhang [52] as follows:

Step 1: Let S be the initial solution, $f(S)$ its objective function value. Set $S^* = S, i = 1, j = i + 1 = 2$.

Step 2: Consider exchange result $S^{'}$ such that $f(S^{'}) < f(S^*)$. Set $S^* = S^{'}$. if $j < n$ repeat *step 2*. Otherwise set $i = i + 1$ and $j = i + 1$. if $i < n$ repeat *step 2*, otherwise go to *step 3*.

Step 3: if $S \neq S^*$ set $S = S^*, i = 1, j = i + 1$ and go to *step 2*. Otherwise output best solution S and terminate the process

2.4 Artificial Bee Algorithm

The ABC algorithm was originally developed by Karaboga [48], for the purpose of multivariable multi-modal continuous functions optimisation. Since then, many variations to the original ABC algorithm were created and employed in solving different types of problems (constrained optimisation [49], multi-objective optimisation [60], combinatorial optimisation ([40], [43], [85], [113])).

In the classification of optimisation heuristics, the Artificial Bee Colony belongs to the group of population, swarm intelligence based stochastic algorithms. Along with other algorithms of this category (such as ACO [29], PSO [51], SOMA ([129],[24]) or Artificial Immune Algorithm (AIA) [4]), ABC searches for optimal solution employing certain number of intelligent agents, who search independently, but also share the information on the system, in order to achieve more efficient behaviour. Amongst the advantages of such approach is inherent parallelism and scalability – it is possible to assign different subgroups of agents to different computational resources.

2.4.1 Description

As well as many other population based metaheuristics, ABC takes inspiration from nature. Namely, from the intelligent behaviour of foraging honey bee swarm. From this inspiration stems also the nomenclature. Several solutions, called food sources, form the population. These solutions are exploited by employed bees. Each employed bee tries to improve one solution at a time (which symbolises extracting a nectar from a food source). Onlooker bees are waiting in the hive to make a decision where to look for a food source. When employed bee arrives, it performs a specific dance, which points the onlooker bee to the direction of a food source. The onlooker bee then determines, based on observation of returning employed bees, where to go. The algorithm emulates this by evaluating the cost function for each solution generated by employed bee. The onlookers choose a solution to improve, based on probability given by relative amount of nectar (cost function value relative to sum of all solutions costs). Finally, if the food source has been exhausted, a scout bee will try to find a new one, unrelated to the previous. Both employed bees and onlooker bees perform exploitation, whereas scout bees responsibility is the exploration of solutions space [47].

The overall algorithm therefore consists of initialisation of population, and three phases performed in succession iteratively, until the predefined terminating criterion is met: employed bee phase, onlooker bee phase, and eventually scout bee phase, as outlined above. The top-level pseudocode of ABC is given in Algorithm 1.

```
1 initialize
2 repeat
3 |   send employed bees to food sources
4 |   send onlooker bees to selected food sources
5 |   send scout bees to search for new food sources
6 |   memorize best solution
7 until terminating criterion met
```

Algorithm 1: ABC top-level pseudocode

The solution in continuous space is represented as a numerical vector of dimension D, whose elements are values of optimised parameters. The population is formed by FS such solutions. The quantity associated with each solution, before it is considered exhausted (a number of trials for bees to try and improve the solution) is called limit. Limit, FS and terminating criterion are the only controlling parameters of the ABC algorithm. Dimension D is given by a definition of optimised function.

2.4.2 Initialisation

Unless some preliminary information on the system is available, the population of solutions is initialised randomly as follows:

$$\mathbf{x}_{i,j} = L_j + (U_j - L_j) \cdot r \tag{1}$$

Where $\mathbf{x}_{i,j}$ denotes j-th element of i-th vector, $j \in \{1, 2, \ldots, D\}$, $i \in \{1, 2, \ldots, FS\}$. L_j and U_j is lower, respective upper bound of j-th dimension of the search space (minimal, respective maximal value allowed for the j-th element of a vector), and r represents a random number in range $[0, 1]$.

2.4.3 Employed bee phase

In employed bee phase, as mentioned earlier, each employed bee i tries to improve a single solution $\mathbf{x}_i$, by performing an operation on it, according to Equation (2). The newly created candidate solution $\mathbf{n}_i$ is evaluated, and greedy selection is applied – if a solution $\mathbf{n}_i$ is better than or equal to previous $\mathbf{x}_i$, the previous one is replaced in the population.

$$\mathbf{n}_{i,j} = \mathbf{x}_{i,j} + (\mathbf{x}_{i,j} - \mathbf{x}_{k,j}) \cdot \mathbf{r}_{i,j} \tag{2}$$

Where $\mathbf{x}_i$ is previous solution in the population, $\mathbf{n}_i$ is the candidate solution, $i \in \{1, 2, \ldots, FS\}$, index j is randomly chosen in range $[1, D]$, index k is randomly selected from range $[1, FS]$ so that $k \neq i$ and $\mathbf{r}_{i,j}$ is a random number in range $[-1, 1]$.

2.4.4 Onlooker bee phase

In onlooker bee phase, a selection from all the food sources (solutions) $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{FS}\}$ is performed, based on the probability of each solution, $p_i, i \in \{1, 2, \ldots, FS\}$, defined in Equation (3). The selected solution is modified using Equation (2) and the greedy selection is again performed between the original and the modified solution, to determine the winning solution to be left in the population, in the same way as described in Section 2.4.3.

$$p_i = \frac{f_i}{\sum_{k=1}^{FS} f_k} \tag{3}$$

In Equation (3), p_i represents probability, f_i is fitness of $i{-}th$ solution, $i \in \{1, 2, \ldots, FS\}$. FS is total number of solutions in the population.

2.4.5 Scout bee phase

As mentioned earlier, each solution has a limit of attempts to improve, associated with it. It is being incremented in both employed bee and onlooker bee phase, in case the solution fails to improve (the newly created solution is worse and therefore not selected). If this limit is exceeded, a scout bee generates new, random solution, according to Equation (1), which replaces the exhausted one. In each iteration of the original ABC algorithm, at most one scout bee is released. The increase in the number of scout bees encourages exploration (escaping from local optima and searching in the global space), but also reduces the possibility of exploitation of good solutions already found, since these are removed from the population by this process [48].

2.5 Discrete Artificial Bee Algorithm

The DABC algorithm by [113] is a modification to the ABC algorithm for solving combinatorial optimisation problems. As opposed to the continuous optimisation, a solution in combinatorial optimisation is naturally encoded as a permutation of elements. Several approaches are possible for transforming continuous optimisation algorithm for solving combinatorial problems, as described for example in [80].

DABC replaces the neighbourhood generation operation of original ABC by a set of operations which transform one permutation to another, thus completely avoiding generation of infeasible solutions. These operations (4 operations altogether, described in section 2.5.2) are based on swapping two random elements, or inserting an element into permutation.

Each of the operations explores a different type of neighbourhood of a solution, furthermore, each of them is suitable for exploring solution space of different problems. In order to maximally adapt to given problem and explore the neighbourhood in an efficient way, all 4 defined operations are used in an adaptive mechanism. This adaptive strategy, described in section 2.5.3, decides which operation to use, partly depending on the list of previous successful operations (those which produced improved solution), partly randomly.

To enhance the exploitation of solution space, DABC contains embedded local search, described in section 2.5.4.

2.5.1 Solution as permutation

As mentioned earlier, a solution is represented as a D-dimensional permutation of elements, as shown in Equation (4). The objective of optimisation is to find the least cost permutation. π_i represents i-th solution, $\pi_{i,j}$ the element at j-th position in the permutation.

$$\pi_i = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,D}\} \tag{4}$$

2.5.2 Operations

As mentioned earlier, DABC makes use of 4 operations: Insert, Swap, $2\times$ Insert, $2\times$ Swap, defined as follows:

Insert removes a randomly selected permutation element from it's position j, reinserts at different randomly selected position k:

$$\begin{aligned} \pi^{(i)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_k, \pi_{k+1}, \ldots, \pi_d) \\ \pi^{(i+1)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_k, \pi_j, \pi_{k+1}, \ldots, \pi_d) \end{aligned} \tag{5}$$

Swap exchanges 2 different randomly selected elements j, k:

$$\begin{aligned} \pi^{(i)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_k, \pi_{k+1}, \ldots, \pi_d) \\ \pi^{(i+1)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_k, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_j, \pi_{k+1}, \ldots, \pi_d) \end{aligned} \tag{6}$$

$2\times$ Insert performs 2 successive Insert operations, as defined above. $2\times$ Swap performs 2 successive Swap operations. Which operation is applied to modify given solution depends on adaptive strategy list.

2.5.3 Adaptive Strategy

Adaptive strategy, as described in [113], maintains two lists of operations, behaving like stacks – list of available operations NL, and list of previously successful operations WNL. On the beginning of the optimisation, the NL is filled with randomly selected operations (4 available operators). Before a solution is modified, operation is taken from the top of NL. If the solution improves upon the previous one, the operation is inserted into WNL. If a NL is empty, part of it is refilled from operations stored in WNL, the rest is refilled again with randomly selected operations. If WNL is empty, the last NL is used again.

2.5.4 Local Search

Local search is embedded within DABC. In employed bee phase, after a new successful solution is generated, the local search is performed with probability PL, in order to further enhance it. The pseudocode is given in Algorithm 2.

2.5.5 Algorithm structure

DABC has similar structure to original ABC algorithm (pseudocode is shown in Algorithm 3). It consists of initialisation, and several iterations of bee phases performed sequentially, until stopping criterion is met. Unless some preliminary information on the problem solution space is known, the population is initialized as a set of random permutations.

2.5.6 Parameters

There are 7 control parameters of DABC, 2 of which are only for usage in adaptive strategy. FS is a number of solutions or food sources in the population, and also the number of employed bees and onlooker bees. $Limit$ is a maximum number of unsuccessful trials to improve the solution, before it is abandoned. $Loop_{max}$ defines a number of iterations in local search, PL is a probability of local search to happen. Stopping criterion in this variant is specified as a number of iterations of DABC algorithm to perform, T. The adaptive strategy requires two parameters, NL_L, WNL_L, defining length of NL list and WNL list, respectively [113]. The range and recommended values of parameters are described in Table 2.

Input: $\pi, fitness, last\_operation$
begin select operation:
  if $last\_operation \in \{insert, 2 \times insert\}$ **then**
    $operation \leftarrow swap$
  else
    $operation \leftarrow insert$
  end
end
begin
  $\pi^{(1)} \leftarrow \pi$
  $fitness^{(1)} \leftarrow fitness$
  for *i = 1* **to** $loop_{max}$ **do**
    $\pi_c \leftarrow operation(\pi^{(i)})$
    $fitness_c \leftarrow$ evaluate π_c
    if $fitness_c$ *better than or equal to* $fitness^{(i)}$ **then**
      $\pi^{(i+1)} \leftarrow \pi_c$
      $fitness^{(i+1)} \leftarrow fitness_c$
    else
      $\pi^{(i+1)} \leftarrow \pi^{(i)}$
      $fitness^{(i+1)} \leftarrow fitness^{(i)}$
    end
  end
  $\pi \leftarrow \pi^{(i)}$
  $fitness \leftarrow fitness^{(i)}$
end

Algorithm 2: Local Search

begin initialize:
    generate food sources as random permutations
    evaluate food sources
    randomly fill NL
end
repeat
    begin 1.send employed bees to food sources:
        foreach *food source* π_i **do**
            /* get operation from Adaptive strategy list: */
            $operation \leftarrow get\_operation(NL)$
            $\pi_i^{(c)} \leftarrow operation(\pi_i)$
            $fitness_i^{(c)} \leftarrow$ evaluate $\pi_i^{(c)}$
            if $fitness_i^{(c)}$ *better than or equal to* $fitness_i$ **then**
                local search on $\pi_i^{(c)}$ with probability PL
                $\pi_i \leftarrow \pi_i^{(c)}$
                $fitness_i \leftarrow fitness_i^{(c)}$
                update adaptive strategy WNL
            end
            update $limit_i$
        end
    end
    begin 2.let onlooker bees select food sources:
        foreach *onlooker bee* **do**
            randomly pick two food sources π_{r_1}, π_{r_2}
            $\pi_s \leftarrow$ better of π_{r_1}, π_{r_2}
            /* get operation from Adaptive strategy list */
            $operation \leftarrow get\_operation(NL)$
            $\pi_s^{(c)} \leftarrow operation(\pi_s)$
            $fitness_s^{(c)} \leftarrow$ evaluate $\pi_s^{(c)}$
            if $fitness_s^{(c)}$ *better than or equal to* $fitness_s$ **then**
                $\pi_s \leftarrow \pi_s^{(c)}$
                $fitness_s \leftarrow fitness_s^{(c)}$
                update adaptive strategy WNL
            end
            update $limit_s$
        end
    end
    begin 3.send the scout to search new food source:
        /* select a solution which was not successfully improved given number of trials: */
        $\pi_s \leftarrow \pi_i$, where: $limit_i \geq Limit$
        $\pi_s \leftarrow$ perform 3 insert operations on π_{best}
        $fitness_s \leftarrow$ evaluate π_s
        $limit_s \leftarrow 0$
    end
    begin 4.memorize best food source:
        $\pi_{best} \leftarrow \pi_i$, where: $\forall k, k \neq i : fitness_i$ better than or equal to $fitness_k$
        $fitness_{best} \leftarrow fitness_i$
    end
until *stopping criterion met*

Algorithm 3: DABC pseudocode

Table 2: Parameter values

Parameter	Recommended value	Range
FS	30	[5, ∞]
$Limit$	50	[1, ∞]
$Loop_{max}$	200	[1, n]
PL	0.2	[0, 1]
T	100	[1, n]
NL_L	20	[1, n]
WNL_L	0.75 NL_L	[1, n]

2.6 Chaos Systems

The term chaos describes the complex behaviour of simple dynamical systems. When casually observed, this behaviour may seem erratic and somewhat random, however, these systems are deterministic. The precise description of their future behaviour is well known, given by the trajectory on the map. This aperiodic non-repeating behaviour of chaotic systems has opened up the possibility to employ the chaotic sequences in place of pseudo-random sequences, as discussed in Section 2.6.1. Generally, four branches of chaotic systems exist, which are the dissipative systems, fractals, dissipative and high-dimensional systems and conservative systems. The systems of interest in this line of research are the discrete dissipative systems. The ones explored in this thesis are introduced in Section 2.6.2.

2.6.1 Chaos pseudo-random number generators applied to Evolutionary Algorithms

One of the main pillars of evolutionary algorithms (EA's) is their reliance on randomness or stochasticity, which is used to spark a path towards a desired goal. The current norm is the use of pseudo-random number generators (PRNG); a structured sequence of mathematical formulation which tries to yield a generally optimal distribution of numbers within a specified range. Of these, the Mersenne Twister is the most famous and widely used [70], [69].

Part of this thesis concentrates on the generation of chaotic sequences, which are then used as chaos pseudo-random number generators (CPRNG's) in an EA. The afore introduced properties of the chaotic systems are employed in place of PRNG embedded in an evolutionary algorithm. The objective is to analyse different chaotic systems, which in this case are the discrete dissipative systems, and to find out, which of these improve the application of EA's.

A mathematical description of the connection between chaotic systems and random number generators has been given by [41]. In this paper, a strong linkage has been shown between the Lehmer generator [58] and the simple chaos dynamical system of Bernoulli shift [82]. The hidden periodicity of chaos system and its dependence on numerical system has been shown by [128]. A chaotic piecewise-linear one dimensional (PL1D) map has been utilised as a chaotic random number generator in [106]. The construction of the chaos random number system is based on the exploitation of the double nature of chaos, deterministic in microscopic space and by its defining equations, and random in macroscopic space. This new system is mathematically proven to overcome the major drawbacks of classical random number systems, which are its reliance on the assumed randomness of a physical process, inability to analyse and optimise the random number generator, inability to compute probabilities and entropy of the random number generator, and inconclusiveness of statistical tests. A family of enhanced CPRNG's has been developed by [65], where the main impetus is the generation of very long series of pseudo-random numbers. This is accomplished through what is called the ultra weak coupling of chaotic systems, such as the Tent Map, which is enhanced in order to conceal

the chaotic genuine function [66]. Recently, the very notion of using CPRNG's in EA's has been explored by [127].

Current literature contains a number of research devoted to certainty, ergodicity and the stochastic property of chaotic systems. Recently, chaotic sequences have been adopted instead of random sequences with improved results. The choice of chaotic sequences is justified theoretically by their unpredictability, i.e. by their spread-spectrum characteristics, non-periodic, complex temporal behaviour, and ergodic properties [81].

A number of EA's have been improved using chaos systems as random number generators during the past few years. Genetic Algorithms (GA) have been improved by chaos to solve multi-objective optimisation problems in [118] and [67] and tourism demand forecasting [42], whereas the Firefly algorithm has been embedded with chaos map in [50]. Differential Evolution (DE) has been improved with chaos to solve the support vector regression machine problem [61], dynamic economic dispatch for wind-thermal power systems [87], loudspeaker design problem [13], optimisation of the batch reactor [101], hydrothermal scheduling [126] and PID control problem [25].

The largest group of chaos based literature is on Particle Swarm Optimisation (PSO). Many variants of chaos embedded PSO exist, some of them are novel creations comprising different chaotic approaches in basic PSO design ([123], [92], [102], [90], [1]), accelerated chaos [34] and hybrid approaches [12] amongst others.

Applications of chaos based PSO include pattern synthesis of antenna arrays [119], image matching [64], power system stabiliser design [30], constrained predictive control [44], global numerical optimisation [12], optimisation of heat exchangers [68], reactive power optimisation [59], network intrusion detection [131], PID Controller design [91] and parameters selection [122].

Some of the other algorithm employing chaos are the Discrete Self-Organising Migrating Algorithm (DSOMA) which has been used to solve the lot-streaming problem [23] and a new artificial emotion based chaos algorithm [125].

Part of this thesis describes the application of CPRNGs to the Discrete Artificial Bee Colony algorithm, resulting in chaos driven Discrete Artificial Bee Colony (CDABC) algorithm, introduced in [71], [74] and [75]. This algorithm is described in Section 3.3.

2.6.2 Chaos Maps

The most interesting chaotic systems, which can be utilised as CPRNG are discrete dissipative chaotic maps, as stated earlier in the text. These maps have the general description of being a linear set of equations, easily formulated, with a fine grain over the solution landscape. This last attribute allows the parsing of unique values over a period of the chaotic oscillation. In total, nine unique chaotic systems were considered for this experiment. The following sections describe the different systems. All operating parameters were obtained from [105].

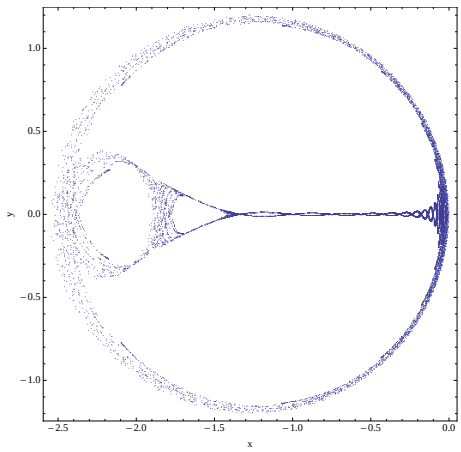

Figure 1: Chaotic Burgers map 2D plot

2.6.3 Arnold's Cat Map

The Arnold's cat map is a two dimensional discrete chaotic map, which is a torus into itself. The equations are given in (7). The parameter of k = 2.0.

$$\begin{aligned} X_{n+1} &= X_n + Y_n \cdot (\mathrm{mod} 1) \\ Y_{n+1} &= X_n + k \cdot Y_n \cdot (\mathrm{mod} 1) \end{aligned} \tag{7}$$

2.6.4 Burgers Map

The Burgers mapping is a discretisation of a pair of coupled differential equations which were used by Burgers [10] to illustrate the relevance of the concept of bifurcation to the study of hydrodynamic flows. It has been numerically shown to produce a much richer set of dynamic patterns than those observed in continuous case [120]. The equations are given in Equations (8) and (9).

$$X_{n+1} = aX_n - Y_n^2 \tag{8}$$

$$Y_{n+1} = bY_n + X_nY_n \tag{9}$$

The operating parameters are a = 0.75 and b = 1.75 with the initial conditions being X_0 = -0.1 and Y_0 = 0.1. As in the previous cases the frequency plot of the real and integer values is given in Figure 2, whereas the x and y values plots are given in Figure 3.

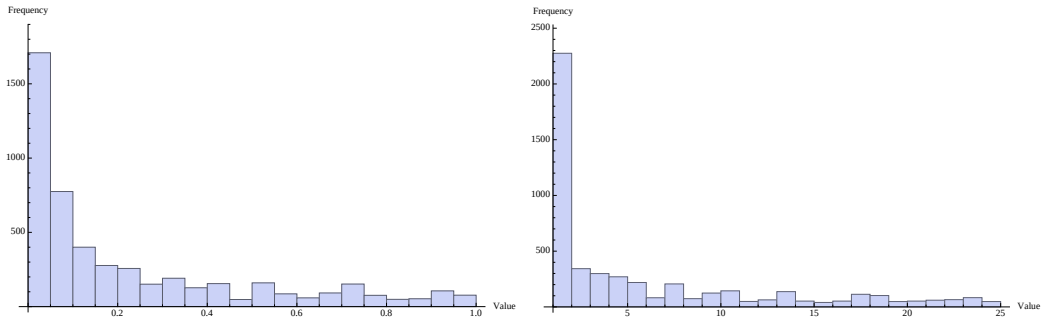

Figure 2: Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Burgers map, 5000 samples.

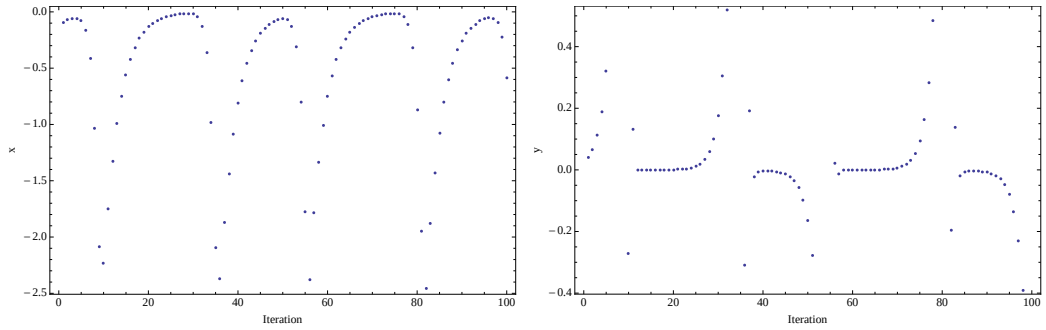

Figure 3: Iteration of the Burgers map for the x and y values in point-plot

2.6.5 Delayed Logistic

The Delayed Logistic map is a dissipative map with a smooth invariant circle interspersed among parameter intervals for which the attractor appears to be strange [3]. This phenomena has given rise to its application in population growth models. The equations of the Delayed Logistic are given in Equations (10) and (11).

$$X_{n+1} = AX_n\left(1 - Y_n\right) \tag{10}$$

$$Y_{n+1} = X_n \tag{11}$$

The operating parameters are $A = 2.27$ and the initial conditions are X_0 = 0.001 and Y_0 = 0.001. The real and integer values histogram in given in Figure 5, and the x and y values plots in Figure 6.

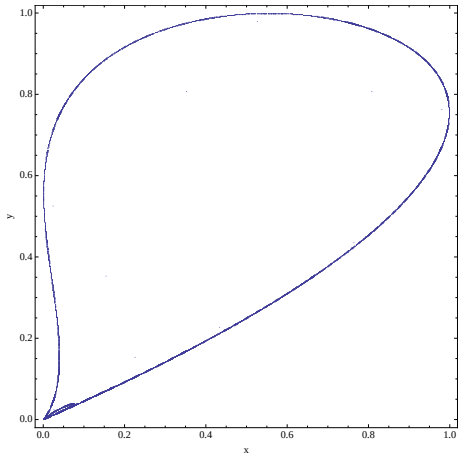

Figure 4: Chaotic Delayed Logistic map 2D plot

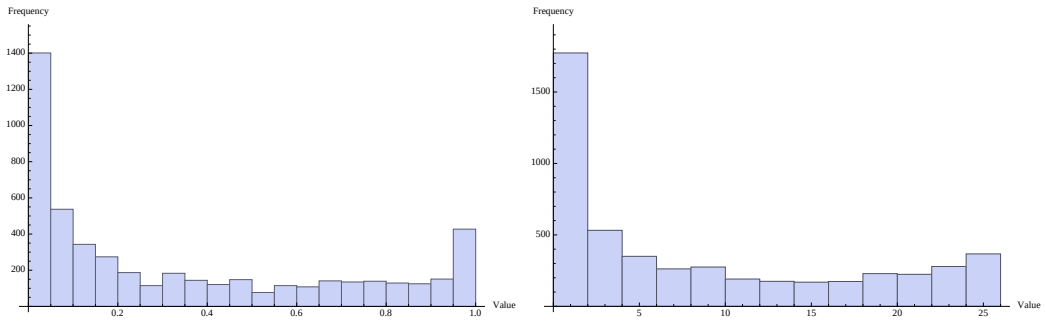

Figure 5: Histogram of the distribution of real numbers transferred into the range $\langle 0, 1 \rangle$ and integer numbers in the range $\langle 1, 25 \rangle$ generated by means of the chaotic Delayed Logistic map, 5000 samples.

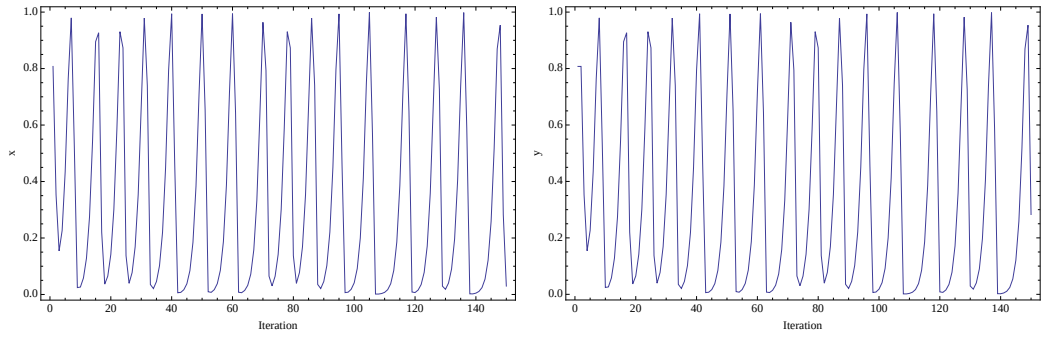

Figure 6: Iteration of the Delayed Logistic map for the x and y values in line-plots

2.6.6 Dissipative Standard Map

The Dissipative Standard Map is a two-dimensional chaotic system. The equation is given in (12) and the operating parameters are β = 0.1 and k = 8.8.

$$\begin{aligned} X_{n+1} &= X_n + Y_{n-1} \cdot (\mathrm{mod} 2\pi) \\ Y_{n+1} &= (\beta \cdot Y_n) + (k \cdot \sin X_n \, (\mathrm{mod} 2\pi)) \end{aligned} \tag{12}$$

2.6.7 Henon Map

The Henon map is a discrete-time dynamical system, which was introduced as a simplified model of the Poincare map for the Lorenz system. The equation is given in (13) and the control parameters are α = 1.4 and β = 0.3.

$$\begin{aligned} X_{n+1} &= \alpha - X_n^2 + (\beta \cdot Y_n) \\ Y_{n+1} &= X_n \end{aligned} \tag{13}$$

2.6.8 Ikeda Map

The Ikeda map is a discrete-time dynamical system derived as a model of light going around across a nonlinear optical resonator. A 2D real example of the Ikeda map is given in equation (14). The operating parameters are α = 0.75, β = 1.75, γ = 1 and μ = 0.9.

$$\begin{aligned} X_{n+1} &= \gamma + \mu \cdot ((X_n \cdot \cos\phi) - (Y_n \cdot \sin\phi)) \\ Y_{n+1} &= \mu \cdot ((X_n \cdot \sin\phi) + (Y_n \cdot \cos\phi)) \\ \phi &= \beta - \tfrac{\alpha}{(1+X_n^2+Y_n^2)} \end{aligned} \tag{14}$$

2.6.9 Lozi Map

The Lozi map is a two-dimensional piecewise linear map whose dynamics are similar to those of the better known Henon map and it admits strange attractors.

The advantage of the Lozi map is that one can compute every relevant parameter exactly, due to the linearity of the map, and the successful control can be demonstrated rigorously.

The Lozi map equations are given in Equations (15) and (16).

$$X_{n+1} = 1 - a\,|X_n| + bY_n \tag{15}$$

$$Y_{n+1} = X_n \tag{16}$$

The parameters used are $a = 1.7$ and $b = 0.5$ as suggested in [105] and the initial conditions are X_0 = -0.1 and Y_0 = 0.1. The real number and integer number plots for a sample iteration is given in Figure 8 and the *x* and *y* value 2D plots is given in Figure 9. The figures presented of the chaotic maps are referenced from [103].

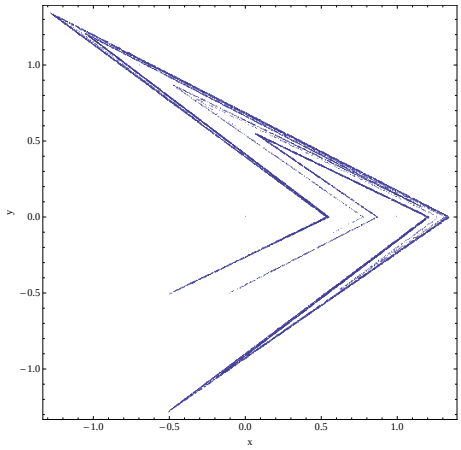

Figure 7: Chaotic Lozi map 2D plot

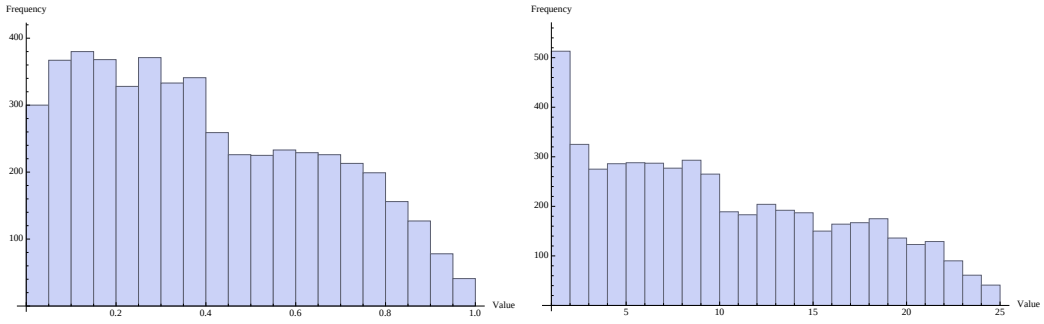

Figure 8: Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Lozi map, 5000 samples.

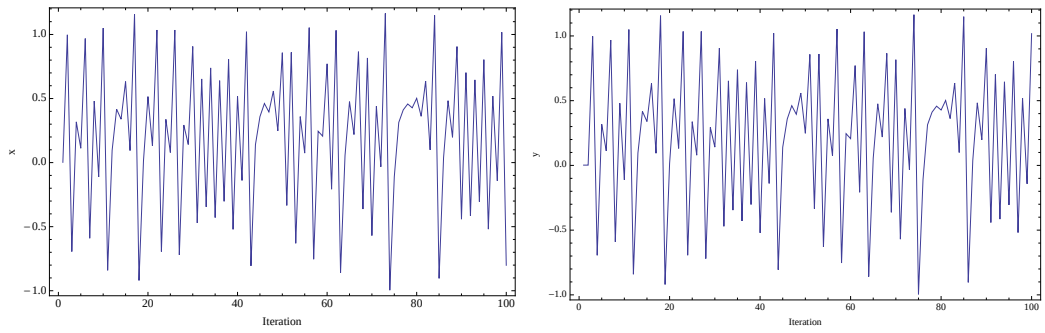

Figure 9: Iteration of the Lozi map for the x and y values in line-plot.

2.6.10 Sinai Map

The Sinai map is a simple two-dimensional discrete system similar to the Arnold's Cat map. The equation is given in (17) and the control parameter is δ = 0.1.

$$\begin{aligned} X_{n+1} &= X_n + Y_n + (\delta \cdot \cos 2\pi \cdot Y_n \cdot (\mathrm{mod} 1)) \\ Y_{n+1} &= X_n + 2 \cdot Y_n \cdot (\mathrm{mod} 1) \end{aligned} \tag{17}$$

2.6.11 Tinkerbell Map

The Tinkerbell is yet another Dissipative map, which has been proven to be chaotic [2] and studied extensively for its unique chaotic attractor. The equations of the Tinkerbell is given in Equations (18) and (19).

$$X_{n+1} = X_n^2 - Y_n^2 + aX_n + bY_n \tag{18}$$

$$Y_{n+1} = 2X_nY_n + cX_n + dY_n \tag{19}$$

The usual operating parameters for Tinkerbell are a = 0.9, b = -0.6, c = 2 and d = 0.5. The initial conditions are X_0 = 0 and Y_0 = 0.5. The real and integer frequency plots are given in Figure 11, whereas the x and y values are given in Figure 12.

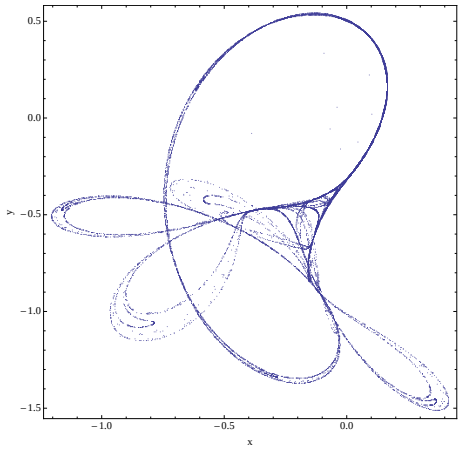

Figure 10: Chaotic Tinkerbell map 2D plot

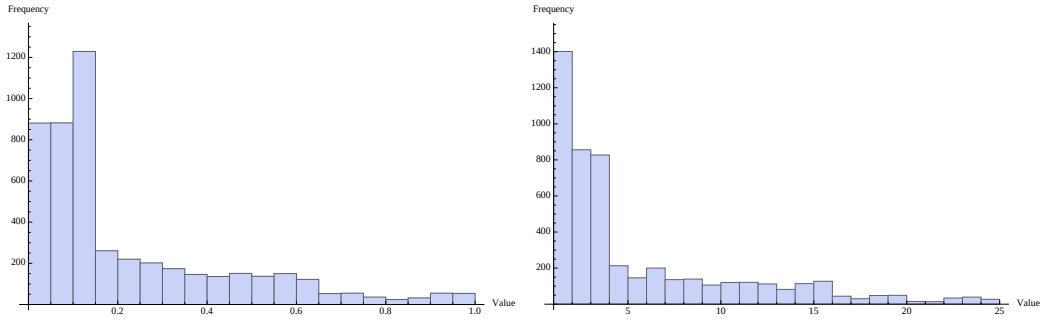

Figure 11: Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Tinkerbell map, 5000 samples.

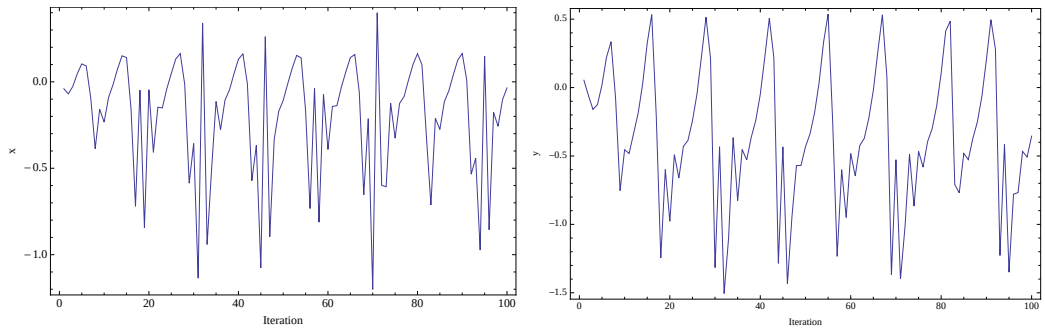

Figure 12: Iteration of the Tinkerbell map for the x and y values in line-plot

2.7 Complex Networks

Many real world systems, both man-made and natural, form a network. Amongst many examples are the communication systems (Internet, World-wide web), transportation systems, neural networks, or protein interaction networks, social interaction structures (social networks), etc. These networks are often too large to describe completely, hence the local description is used instead, using probabilistic expression of the local properties in place of the exact ones, considering the random graphs [8],[116].

2.7.1 Complex Network Analysis

Many evolutionary algorithms based on the population of solutions form a complex network in terms of exchange of information amongst the individuals throughout the search for the optimum. As an example, SOMA [27] and DE [26] were analysed from this point of view, and proven to form the complex network in the course of the algorithm iterations. In this work, the information flow amongst the solutions of the population in the ABC algorithm is explored, and the knowledge of the complex network structure and properties is subsequently used to moderate the search.

The ABC forms a complex network during the course of the iterations, which was empirically validated. This complex network was recorded as a weighted adjacency matrix, representing the directed graph, where the vertices represent the individuals, while oriented edges stand for the information flow between them, with weights representing the number of successful improvements of the target individual using the information from the source individual.

The complex network structure can be analysed by many different techniques and measures. In order to measure the influence and importance of the individual nodes to the whole network, the centrality of the vertices was chosen as a natural way to accomplish this goal.

2.7.2 Centrality

Centrality of vertices is a means of analysis of complex network by identifying the most important or influential nodes in the graph. Several centrality measures exist, the most common being the *Degree centrality*, *Closeness* and *Betweenness*, which measure the direct connections of the node, accessibility of the node and how much the node is intermediary between other nodes, respectively.

Degree Centrality Degree centrality of the node is the number of the edges incident with the node. For a directed graph, the number of only incoming (in-degree), outgoing (out-degree) or both edge types can be considered. In a weighted network, the degree of the node has been extended as a sum of the incident edges weights. The degree centrality

measures the local connections of the node. The unweighted degree centrality of node i is defined as follows:

$$C_D(i) = \sum_{j=1}^{n} a_{ij} \tag{20}$$

Where n is the total number of nodes, a_{ij} is 1 if an edge exists between nodes i and j, 0 if it doesn't exist [117].

Weighted degree centrality (strength) of node i:

$$C_S(i) = \sum_{j=1}^{n} a_{ij} w_{ij} \tag{21}$$

The meaning of n and a_{ij} is the same as for the unweighted degree, w is weight of the edge between i and j [8].

Closeness Centrality The closeness of a node measures it's accessibility - how close the node is along the shortest path to all the other nodes, on average. It is defined as the inverse of the average shortest distance from the node to all other nodes. The following equation defines the closeness centrality of the node i:

$$C_C(i) = \frac{n-1}{\sum_{j \in \{1..N\}, j \neq i} d_{ij}} \tag{22}$$

Where d_{ij} is the shortest path between i and j [117].

Betweenness Centrality The betweenness measures how much the node lies between other nodes in the network. The nodes with high betweenness centrality are likely to serve as mediators in information exchange between other nodes, and therefore can be seen as more important. The betweenness of a node is a ratio of the shortest paths between other node pairs passing through the node. Betweenness centrality of node i is given by the following equation:

$$C_B(i) = \sum_{k,j \in \{1..N\}, k \neq j \neq i} \frac{\sigma_{k,j}(i)}{\sigma_{k,j}} \tag{23}$$

Where $\sigma_{k,j}$ is the number of the shortest paths between k and j, and $\sigma_{k,j}(i)$ is the number of shortest paths between k and j that pass through i [117].

2.8 CUDA

The GPGPU (general purpose GPU computing) has been in the focus of many researches ever since the GPUs performance rapid increase, owing to the increasing demand for powerful hardware capable of rendering ever more demanding game graphics, along with the increasing programmability of the GPU. However, the first GPU really supporting scientific computations was developed several years later in 2006 by NVIDIA, whose architecture CUDA (Compute Unified Device Architecture) eventually enabled programming GPUs by means of small set of extensions to C/C++ language, as opposed to former necessity of using graphic programming languages and primitives ([97], [53])

As mentioned earlier, CUDA-enabled GPUs are programmed using extensions to C/C++ language, CUDA C language. Furthermore, several different languages, APIs or CUDA accelerated libraries are supported. CUDA programming model is data parallel and widely scalable. The computational task is divided between multicore CPU and manycore GPU with separate memory spaces and different properties, strengths and weaknesses – so called heterogenous programming. Compute intensive data parallel tasks are offloaded to the GPU, while tasks requiring sophisticated flow control are executed sequentially on the CPU.

At the heart of CUDA programming model are three key abstractions: *thread hierarchy*, *memory hierarchy* and *synchronization*, providing coarse grained parallelism (*blocks* in *grid*), and fine grained parallelism (*threads* in *block*, which can communicate and be synchronized).

Thread hierarchy The data parallel task is implemented using special function called *kernel*, whose code is executed in parallel by threads. *Threads* are organized into *blocks*, blocks of threads are organized into *grids*. Whereas threads in each block can communicate by means of *shared memory* and *synchronization* function, threads between different blocks are completely independent of one another. Each thread within block, as well as each block within grid, is distinguishable by *threadId*, respectively *blockId*. This enables each particular thread to operate on different data element in the *global memory*.

Memory hierarchy CUDA application can make use of different memory types. Memories differ in size and speed, as well as supported effective access patterns. Some of them are cached, some are read-only. The design of effective memory usage is one of the key issues of CUDA program performance. So called *global memory* is shared by all blocks, as well as between successive kernel calls, and is relatively slow. Much faster *shared memory* is accessible by all threads within one block. The fastest *register memory* is used to store local variables for one thread, which are not visible to any other thread. However, for big data structures or if the total amount of memory needed for local variables by all threads within block exceeds the registers capacity, the slower *local memory* must be used. *Constant memory* can be employed to store data that will not be changed by the kernel code, especially when all threads access the same data element at a time. The *texture memory* is a read-only cache that provides a speed-up for locality in data access by threads [78].

Synchronization As mentioned earlier, blocks in grid provide for coarse grained parallelism, whereas threads in block provide for fine grained parallelism. Threads in a block can be synchronized and share data in the scope of a kernel. The number of threads in a block is however limited both by the CUDA GPU design (max. 1024), and by the memory resources consumed by each thread. This division permits scalability – the blocks are scheduled independently of one another, each of them assigned to one of the GPU's multiprocessors ([77], [78]).

2.9 Combinatorial Optimisation and Scheduling Problems

This section gives the overview of the theory and formulation of the problems to which the combinatorial optimisation methods explored in this thesis have been applied. The combinatorial optimisation is the subset of optimisation problems, generally formulated, where an optimal solution from the finite set of feasible solutions is sought after. They can be defined as seeking a subset $S^* \in \mathbf{S}$, given the collection $\mathbf{S} \subseteq 2^E$ on some finite ground set E; and $c : \mathbf{S} \to \mathbb{R}$; such that S^* maximizes or minimizes c on $\mathbf{S}$ [28].

The problems of interest from the combinatorial optimisation field are the NP-hard problems where the exhaustive search methods are infeasible. Of this category, the quadratic assignment problem and capacitated vehicle routing problem are presented later in the text, in Sections 2.9.1 and 2.9.2.

The special consideration is given to the scheduling problems. The scheduling as a process of decision making is used on daily basis in manufacturing and production systems and service industries. The scheduling problems in general deal with the task of sequencing a collection of jobs in certain machine environment, subject to given constraints, in such a way that one or more performance criteria are optimised. Of the two broad classes of scheduling models, the deterministic and stochastic scheduling, the deterministic model is considered, with the finite collection of jobs to be scheduled, where the exact job data are known in advance. The following text describes the lot streaming flow shop with setup times (Section 2.9.5), permutative flow shop and the flow shop with no-wait constraint (Sections 2.9.3 and 2.9.4) [89].

2.9.1 Quadratic Assignment Problem

The quadratic assignment problem (QAP) is a combinatorial optimisation problem stated for the first time by [55] and is widely regarded as one of the most difficult problems in this class. The objective is to assign ***n*** facilities to ***n*** locations in such a way as to minimise the assignment cost.

The assignment cost is the sum, over all pairs, of the flow between a pair of facilities multiplied by the distance between their assigned locations.

Let *C* and *D* be two $n \times n$ matrices such that $C = [c_{i,j}]$ and $D = [d_{i,j}]$. Consider the set of positive integers $\{1, 2, \ldots, n\}$, and let S_n be the set of permutations of $\{1, 2, \ldots, n\}$. Then the quadratic assignment problem can be defined as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} d_{\pi(i)\pi(j)} \tag{24}$$

over all permutations $\pi \in S_n$. The above formulation is known as the Koopmans-Beckman QAP [55].

Stated in other words, the objective of the quadratic assignment problem with cost matrix *C* and distance matrix *D* is to find the permutation $\pi_0 \in S_n$ that minimises the double summation over all *i*, *j*.

It should be understood that the notation $d_{\pi(i)\pi(j)}$ as used above, refers to permuting the rows and columns of the matrix *D* by some permutation π.

That is, $D^{\pi} = [d^{\pi}{}_{i,j}] = d_{\pi(i)\pi(j)}$, for $1 \leqslant i, j \leqslant n$. In the same manner, given an n-dimensional vector $V = [v_i]$, a permutation of the elements of V by a permutation π will be denoted as $V^{\pi} = [v^{\pi}{}_{i}] = v_{\pi(i)}$.

A number of heuristics have been developed to handle large scale QAP problems; some notable ones being simulated annealing [14], tabu search [107] and the hybrid genetic-tabu search [32].

2.9.2 Capacitated Vehicle Routing Problem

The vehicle routing problem is a well known problem in the field of transportation ([18], [57], [115], [6], [36], [7]). The basics of capacitated vehicle routing problem can be stated as follows [63]. Each vehicle has the same loading capacity, and starts off from only one delivery depot and then routes through customers. All customers have known demands and required service time. Each customer can only be visited by one vehicle, and each vehicle has to return to the depot. The service time unit can be transformed into the distance unit. The loading and traveling distance of each vehicle cannot exceed the loading capacity and the maximum traveling distance of vehicle. The objective of CVRP is to minimize the traveling cost. The capacitated vehicle routing problem can be modeled as a mixed integer programming as follows:

$$\min \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{K=1}^{K} C_{ij} X_{ij}^{k} \tag{25}$$

Subject to:

$$\sum_{i=0}^{N} \sum_{j=0}^{N} X_{ij}^{k} d_i \leq Q^k \quad 1 \leq k \leq K, \tag{26}$$

$$\sum_{i=0}^{N} \sum_{j=0}^{N} X_{ij}^{k} \left(C_{ij} + S_i\right) \leq T^k \quad 1 \leq k \leq K, \tag{27}$$

$$\sum_{j=1}^{N} X_{ijk} = \sum_{j=1}^{N} X_{jik} \leq 1 \ \text{ for } \ i = 0 \ \text{ and } \ k \in \{1, \ldots, k\}, \tag{28}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N} X_{ijk} \leq K \ \text{ for } \ i = 0, \tag{29}$$

where C_{ij} is the cost incurred on customer i to customer j, K the number of vehicles, N the number of customers, the S_i the service time at customer i, Q^k the loading capacity of vehicle k, T^k the maximal traveling (route) distance of vehicle k, d_i the demand at customer i, $X_{ij}^{k} \in 0$ and $1\,(i \neq j; i, j \in 0, 1, \ldots, N)$.

Equation 25 is the objective function of the problem. Equation 26 is the constraint of loading capacity, where $X_{ij}^{K} = 1$ if vehicle k travels from customer i to customer j directly,

and 0 otherwise. Equation 27 is the constraint of maximum traveling distance. Equation 28 makes sure every route starts and ends at the delivery depot. Equation 29 specifies that there are maximum *K* routes going out of the delivery depot.

2.9.3 Permutative Flowshop scheduling problem

In many manufacturing facilities, each job has to undergo a number of operations in given order. The machines which process the jobs are then set up in series and the environment is referred to as flowshop. Moreover, if the jobs cannot skip one another in the queue between machines, i.e. first-in-first-out principle applies, the environment is an instance of permutative flowshop. The optimal schedule is given by the permutation of the order of jobs. If the objective is to minimise the makespan, the sheduling problem is given in the standard notation as:

$$Fm\,|prmu\,|C_{\max}$$

where the $C_{\max}$ denotes the makespan objective, the completion time of the last job on last machine. This problem is proven to be strongly NP-hard. Being one of the basic scheduling problems, it has attracted much attention in the research in past years.

The problem is formulated as follows: Given the order of n jobs to be processed on m machines in series, the processing time $p_{i,j}$ of job j on machine i, and the permutation schedule $j_1, \ldots, j_n$, the completion time of a job j_k on machine i can be computed by the set of recursive equations:

$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \qquad i = 1, \ldots, m \tag{30}$$

$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_l} \qquad k = 1, \ldots, n \tag{31}$$

$$C_{i,j_k} = \max(C_{i-1,j_k}, C_{i,j_{k-1}}) + p_{i,j_k} \qquad i = 2, \ldots, m; k = 2, \ldots, n \tag{32}$$

The makespan of the schedule is given as $C_max = \max(C_1, \ldots, C_n)$ [89].

2.9.4 Flowshop with Zero Intermediate Storage

One of the most challenging and practical scheduling problem in the flowshop class is the one with no storage or stoppage between machines [88]. Consider a flow shop with zero intermediate storage (FSSZIS) subject to different operating procedures. A job, when it goes through the system, is not allowed to *wait* at any machine. For this process, all subsequent machines have to be *idle*, at the completion of the job on a machine upstream. Therefore, the jobs are ***pulled*** down the line by machines which have become idle. This constraint can be also refereed to as the **no-wait** constraint, and minimising the makespan in such a flow shop is referred to as the

$$Fm\,|nwt\,|C_{\max}$$

Among all types of scheduling problems, FSSZIS owns lots of important applications in different industries such as chemical processing [96], food processing [39], concrete ware production [37], and pharmaceutical processing [94] amongst others.

For the computational complexity of the FSSZIS scheduling problem, [35] proves that it is strongly NP-complete. Therefore, only small size instances of this flowshop problem can be solved with reasonable computational time by exact algorithms.

The following notations are used to formulate the FSSZIS problem: assume n as number of jobs to be scheduled, m as the number of machines in the flowshop, $t_{i,j}$ as the processing time for the i^{th} job on the j^{th} machine, $d_{i,k}$ as the minimum delay on the first machine between the start of job i and job k due to the no-wait constraint, $[i]$ as the job processed in position i , $C_{[i]}$ as the completion time of the job processed in position. TFT represents the total flow time, i.e. the sum of flow times of all jobs.

The minimum delay time $d_{i,k}$ and completion time $C_{[i]}$ can be calculated as:

$$\begin{aligned} d_{i,k} &= t_{i,1} + \max_{2\leq j\leq m}\left(\sum_{p=2}^{j} t_{i,p} - \sum_{p=1}^{j-1} t_{k,p}\right) \\ C_{[i]} &= \sum_{j=1}^{m} t_{[1],j}, \\ C_{[i]} &= \sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{j=1}^{m} t_{[1],j}, \qquad i = 2,3,\ldots,n. \end{aligned} \tag{33}$$

All jobs are assumed to be available at time zero, the total flow time can then be given as in (34).

$$\begin{aligned} &TFT = \sum_{i=2}^{n}\left(\sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{j=1}^{m} t_{[1],j}\right) + \sum_{j=1}^{m} t_{[1],j} = \\ &\sum_{i=2}^{n}\sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{i=1}^{n}\sum_{j=1}^{m} t_{[i],j} = \\ &\sum_{i=2}^{n}(n+1-i)d_{[i-1],[i]} + \sum_{i=1}^{n}\sum_{j=1}^{m} t_{i,j} \end{aligned} \tag{34}$$

where $\sum_{i=1}^{n}\sum_{j=1}^{m} t_{i,j}$ is the sum of the processing time of all jobs in all machines [11].

2.9.5 Lot Streaming Problem

The lot-steaming problem with setup time considered in this paper is a subset of the generic flowshop scheduling problem. Whereas, in the permutative flowshop problem, each job n is processed by a single machine m, in a lot-streaming variant, each job is divided into smaller tasks called *lots* (l) [83]. Once the processing of a sub-lot on its preceding machine is completed, it can be transferred to the downstream machine immediately. However, all $l(j)$ sub-lots of job j should be processed continuously as no intermingling or exchanging is allowed. A separable sequence-dependent setup time is necessary for the first sub-lot of each job j before it can be processed on any machine k [84].

Two different cases of the problem are available; the idling and the non-idling case. The idling case is the simpler variant of the problem, where only the schedule of the lots is taken into consideration. A non-idling case on the other hand is more practical. A non-idle case arises when the machine is not allowed to be idle. This is beneficial, especially in the case when a number of machines are in operation, and resources, such as electricity, are wasted. Another practical example is when expensive machinery is employed. Idling of such expensive equipment is often not desired. In this paper, only the non-idling case is considered.

For a detailed description of the lot-streaming problem please refer to [98].

Non-Idling Case The constraint in this case is that at any given time a machine can process only one sub-lot, and each sub-lot can only be assessed individually. Let the processing time of each sub-lot of job j on machine m be $P(m,j)$, and the setup time of job j on machine m, after having processed job j is $s(m,j,j)$, which can also represent the setup time of job j if it is the first job to be proceeded on the machine. The objective is to find a sequence with the optimal sub-lot starting and completion times to minimise the makespan.

The permissible job permutation can be presented as $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, and the earliest start and completion time as $S(m,j,r)$ and $C(m,j,r)$, where r represents the specific sub-lot on job j being processed on machine m.

For the non-idling case, the earliest start time for the first sub-lot is given in equations (35) and (36), where the start time is the maximum of the setup time of the job in the current machine, the completion time of the first sub-lot on the previous machine, and the difference between the completion time of the whole job on the previous machine and the total processing time of the whole job on the preceding machine except the last sub-lot. This ensures that there is no idling time between two adjacent sub-lots. The last two directives of these equations calculate the completion time for the first job.

The subsequent processing times of the following job sequence are given in equations (37) and (38).

$$\begin{aligned} &S(1,\pi_1,1) = s(1,\pi_i,\pi_i) \\ &C(1,\pi_1,l(\pi_1)) = S(1,\pi_1,1) + l(\pi_1) \times P(1,\pi_1) \end{aligned} \tag{35}$$

$$\begin{aligned} S(w,\pi_1,1) = \max \left\{ \begin{array}{l} s(w,\pi_1,\pi_1), S(w-1,\pi_1,1) + \\ \qquad p(w-1,\pi_1), \\ C(k-1,\pi_1,l(\pi_1)) - \\ \qquad (l(\pi_1)-1) \times P(1,\pi_1) \end{array} \right\}, \\ C(w,\pi_1,l(\pi_1)) = S(w,\pi_1,1) + l(\pi_1) \times P(w,\pi_i), \\ w = 2,3,\ldots,m \end{aligned} \tag{36}$$

$$\begin{aligned} &S(1,\pi_1,1) = C(1,\pi_{i-1},l(\pi_{i-1})) + s(1,\pi_{i-1},\pi_i), \\ &C(1,\pi_1,l(\pi_1)) = S(1,\pi_1,1) + l(\pi_1) \times P(1,\pi_1), \\ &\qquad i = 2,3,\ldots,n \end{aligned} \tag{37}$$

$$
\begin{aligned}
S\left(w,\pi_1,1\right) &= \max\left\{\begin{array}{l} S\left(w-1,\pi_i,1\right)+P\left(w-1,\pi_1\right), \\ C\left(w-1,\pi_1,l\left(\pi_1\right)\right)- \\ \quad \left(l\left(\pi_1\right)-1\right)\times P\left(1,\pi_1\right), \\ C\left(w-1,\pi_{i-1},l\left(\pi_{i-1}\right)\right)+ \\ \quad s\left(1,\pi_{i-1},\pi_i\right) \end{array}\right\}, \\
& i=2,3,\ldots,n, \quad w=2,3,\ldots,m \\
C\left(w,\pi_1,l\left(\pi_1\right)\right) &= S\left(w,\pi_1,1\right)+l\left(\pi_1\right)\times P\left(w,\pi_i\right), \\
& i=2,3,\ldots,n, \quad w=2,3,\ldots,m
\end{aligned}
\tag{38}
$$

The makespan for the non-idling case can be then calculated as equation (39).

$$
C_{\max}\left(\pi\right)=C_T\left(m,\pi_n,l\left(\pi_n\right)\right) \tag{39}
$$

The objective of the lot-streaming flow shop scheduling problem with makespan criterion is to find a permutation π^* in the set of all permutations $\prod$. It can be given as in the equation (40) [84].

$$
C_{\max}\left(\pi^*\right)\leq C_{\max}\left(\pi\right),\forall\pi\in\Pi \tag{40}
$$

2.10 Continuous Optimisation Problems

This section gives the brief overview of the selected common test functions used for proving the properties and efficiency of evolutionary algorithms for continuous optimisation. For each test function, the equation and the position of global minimum (if known) is given. The test functions were taken from [130], more information can be found in [121].

2.10.1 Schwefel's function

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} -x_i sin\left(\sqrt{|x_i|}\right) \tag{41}$$

The global minimum value can be obtained by the following equation:

$$f(x_1, \ldots, x_n) = -418.9829n \tag{42}$$

for the arguments $x_i = 420.9687, i = 1, \ldots, n$.

2.10.2 1st De Jong's function

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 \tag{43}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = 0 \tag{44}$$

is at $x_i = 0, i = 1, \ldots, n$.

2.10.3 3rd De Jong's function

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} |x_i| \tag{45}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = 0 \tag{46}$$

is at $x_i = 0, i = 1, \ldots, n$.

2.10.4 4th De Jong's function

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} i x_i^4 \tag{47}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = 0 \tag{48}$$

is at $x_i = 0, i = 1, \ldots, n$.

2.10.5 Rosenbrock (2nd De Jong's function)

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} 100 \left(x_i^2 - x_{i+1}\right)^2 + (1 - x_i)^2 \tag{49}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = 0 \tag{50}$$

is at $x_i = 1, i = 1, \ldots, n$.

2.10.6 Rastrigin

$$f(x_1, \ldots, x_n) = 2n \sum_{i=1}^{n} x_i^2 - 10cos(2\pi x_i) \tag{51}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = -200n \tag{52}$$

is at $x_i = 0, i = 1, \ldots, n$.

2.10.7 Griewangk

$$f(x_1, \ldots, x_n) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{53}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = 0 \tag{54}$$

is at $x_i = 0, i = 1, \ldots, n$.

2.10.8 Sine Envelope Sine Wave

$$f(x_1, \ldots, x_n) = -\sum_{i=1}^{n-1} \left(0.5 + \frac{sin\left(x_i^2 + x_{i+1}^2 - 0.5\right)^2}{\left(1 + 0.001\left(x_i^2 + x_{i+1}^2\right)\right)^2}\right) \tag{55}$$

The global minimum value of:

$$f(x_1, \ldots, x_n) = -1.4915(n - 1) \tag{56}$$

2.10.9 Stretched V Sine Wave

$$f\left(x_1,\ldots,x_n\right)=\sum_{i=1}^{n-1}\left(\sqrt[4]{x_i^2+x_{i+1}^2}sin\left(50\sqrt[10]{x_i^2+x_{i+1}^2}\right)^2+1\right) \tag{57}$$

The global minimum value of:

$$f\left(x_1,\ldots,x_n\right)=0 \tag{58}$$

is at $x_i=0, i=1,\ldots,n$.

2.10.10 Ackley's function I

$$f\left(x_1,\ldots,x_n\right)=\sum_{i=1}^{n-1}\left(\frac{1}{e^5}\sqrt{x_i^2+x_{i+1}^2}+3\left(cos\left(2x_i\right)+sin\left(2x_{i+1}\right)\right)\right) \tag{59}$$

The global minimum value (for $n\geq 3$) of:

$$f\left(x_1,\ldots,x_n\right)=-7.54276-2.91867(n-3) \tag{60}$$

2.10.11 Ackley Two

$$f\left(x_1,\ldots,x_n\right)=\sum_{i=1}^{n-1}\left(20+e-\frac{20}{e^{0.2\sqrt{\frac{x_i^2+x_{i+1}^2}{2}}}}-e^{0.5(cos(2\pi x_i)+cos(2\pi x_{i+1}))}\right) \tag{61}$$

The global minimum value of:

$$f\left(x_1,\ldots,x_n\right)=0 \tag{62}$$

is at $x_i=0, i=1,\ldots,n$.

2.10.12 Egg Holder

$$f\left(x_1,\ldots,x_n\right)=\sum_{i=1}^{n-1}\left(-x_isin\left(\sqrt{|x_i-x_{i+1}-47|}\right)-(x_{i+1}+47)sin\left(\sqrt{|x_{i+1}+47+\frac{x_i}{2}|}\right)\right) \tag{63}$$

2.10.13 Michalewicz

$$f\left(x_1,\ldots,x_n\right)=\sum_{i=1}^{n-1}\left(-1\left(sin\left(x_i\right)sin\left(\frac{x_i^2}{\pi}\right)^{20}+sin\left(x_{i+1}\right)sin\left(\frac{2x_i^2}{\pi}\right)^{20}\right)\right) \tag{64}$$

The global minimum value (for $n>2$) of:

$$f\left(x_1,\ldots,x_n\right)=1.00098(n-2) \tag{65}$$

2.10.14 Masters Cosine Wave

$$f\left(x_1,\ldots,x_n\right)=\sum_{i=1}^{n-1}\left(e^{\frac{-(x_i^2+x_{i+1}^2+0.5x_{i+1}x_i)}{8}}cos\left(4\sqrt{x_i^2+x_{i+1}^2+0.5x_ix_{i+1}}\right)\right) \tag{66}$$

The global minimum value of:

$$f\left(x_1,\ldots,x_n\right)=-1\cdot n \tag{67}$$

is at $x_i=0,i=1,\ldots,n$.

2.10.15 Shekel's Foxhole

$$f\left(x_1,\ldots,x_n\right)=-1\sum_{j=1}^{m}\frac{1}{c_j+\sum_{i=1}^{n}(x_i-a_{j,i})^2} \tag{68}$$

Where recommended value of $m=30$, and constants $c_j;j=1,\ldots,m$ and $a_{j,i};j=1,\ldots,m;i=1,\ldots,n$ are constant numbers fixed in advance.

3 Implementation

This section presents the description of design and implementation of each of the programs. All programs were developed as independent command line applications. Parameters are provided through the combination of command line arguments and configuration files. All source codes in C/C++ and CUDA C can be found on CD, together with short manuals and examples of usage (see appendix A).

3.1 CUDA based NEH

In this section, the parallel version of NEH algorithm is presented. As discussed in section 2.8, the algorithm uses both the CPU and the GPU. For the most applications, it is desirable to maximally reduce the amount of data transferred between CPU and GPU, because PCI Express is relatively slow and excessive transfers can considerably decrease performance. It is obvious, from the analysis of sequential NEH (Section 2.2), that there exists data dependency in each step of the main iteration. Whereas the process of exploring all the variants derived by extending the current subsequence (partial solution) by the next job consists of generating several alternative solutions and their evaluation, where each solution is independent from all the others, and hence can be implemented as parallel, the best partial solution (subsequence) must then be chosen and used as a basis for the next iteration step (for generating the next set of partial solutions), which requires synchronization. From these considerations stems the design of CUDA based NEH algorithm.

The main loop (encompassing step 3 and 4 of the sequential NEH description) runs on the CPU, due to the afore mentioned data dependency. The algorithm flow inside the main loop can be divided into four parts, each of which can be run in parallel. The basic structure of the algorithm can be described as follows:

Main loop: Until full solution found, do:

Step 1: Generate the candidate subsequences (solutions) from the current subsequence.

Step 2: Evaluate candidate subsequences.

Step 3: Find the best (minimal cost) solution amongst all candidates.

Step 4: Update current subsequence.

3.1.1 Generating the candidate subsequences, Evaluation

Evaluating subsequences, i.e. calculating the problem cost function, is parallelized in a simple way – each solution is evaluated by one thread of one block – therefore all costs are calculated in parallel, but no further parallelism is exploited inside the calculation of a single solution cost function, as main objective of this paper is to explore the possibility of parallelisation of the NEH heuristics itself, rather than that of FSS problem, which was done in [16]. To speed up the calculation, shared memory was used. Full

$machines \times jobs$ completion time matrix of FSS makespan problem would have consumed too much memory space, therefore it was reduced to only $2 \times jobs$ (only 1 last machine completion times for all jobs are remembered in each step, the completion time for subsequent machine is stored into 2nd table row, the rows are then swapped before the next step).

Generation of candidate solutions is fully parallel – each block of threads is responsible for generating a single candidate solution, which means performing a single insert on current sequence. The position into which the next job is inserted in the candidate solution corresponds to *blockId*. Each thread in a block shifts subset of solution elements (jobs), using shared memory. The generated candidate solution could then be stored into global memory and then read again and evaluated in a separate kernel. However, as the kernel for generating the solution was relatively small, it was merged with the kernel performing the evaluation to avoid this delay caused by additional write/read operation on the global memory. In the merged kernel, the solution is first generated, then written into global memory. All the threads in a block apart from the first thread are terminated, the remaining thread performs the evaluation of the solution (while it is still in shared memory), taking advantage of a memory with lower latency, finally storing the calculated cost into global memory.

3.1.2 Find the minimal cost solution

The search for the index of the minimal cost in the array of solutions costs is done using a parallel reduction pattern, employing shared memory to store the data being processed. In the beginning, the data is loaded from global into shared memory. In each step, each active thread compares two costs, and stores the smaller of the two costs on the place of the first cost, along with its original index (cost is represented as a structure containing two elements: cost value, and cost index). After each step, the number of active threads is reduced by half. In the end, the first element of the costs array contains the minimal cost found, along with its respective solution index. This pair is then written into global memory.

3.1.3 Update current subsequence, device synchronization

Finally a simple kernel copies the best solution, based on its stored index, into the current solution buffer, and the next step of the main loop can be performed. However, before it can be started, a global CUDA device synchronization is necessary for a big data (for a job schedule size/number of threads in a block of size more than approximately 100, as was empirically confirmed). As each of the kernels consumes some of the GPU resources, it is necessary to wait, until the pending kernels completely finish the execution, and release their resources, otherwise the GPU freezes and unsuccessful kernel launches start to appear. This is done by calling `cudaDeviceSynchronize()` function from the host code, after the Update kernel is launched.

The following figure (13) depicts the memory layout of the afore described code (without FSS input data, for the current subsequence size 2, full schedule size 4. The data fields

not used in the current step are grayed out). The candidate solutions are stored in one global memory 1D array, which conceptually represents 2D array, wherein each row contains one candidate schedule. The respective costs are stored in a separate array. The FSS problem input data (processing times of each job on each machine) are stored in the similar fashion in global memory (because of its large size).

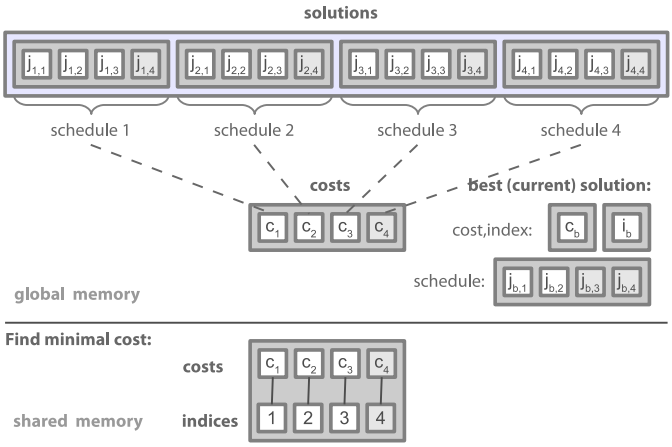

Figure 13: CUDA based NEH memory layout

This implementation is expected to provide in each step the speedup proportional to the number of solutions generated.

3.2 CUDA based 2-opt algorithm

This section presents the parallel CUDA based version of 2-opt algorithm for permutative flowshop with makespan criterion problem. Before coming to the parallel implementation description however, the more detailed pseudocode of sequential version is provided in Algorithm 4, in order to enable better understanding of the CUDA version design.

```
   Input:
   S : initial solution
1
   // number of jobs
2  N ← Size(S)
   // objective function value of S
3  f_S ← f(S)
   // temporary solution memory
4  T ← S
5  while ImprovementFound do
6  |  ImprovementFound ← False
7  |  for i=1 to N-1 do
8  |  |  for j=i+1 to N do
9  |  |  |  T ← Swap(T,i,j)
10 |  |  |  f_T ← f(T)
11 |  |  |  if f_T < f_S then
12 |  |  |  |  S ← T
13 |  |  |  |  f_S ← f_T
14 |  |  |  |  ImprovementFound ← True
15 |  |  |  |  break(2)
16 |  |  |  end
17 |  |  |  T ← Swap(T,j,i)
18 |  |  end
19 |  end
20 end
21 return S
```

Algorithm 4: 2-opt sequential version. The *Swap(T,j,i)* procedure swaps $j-$th and $i-$th job of schedule T

As can be seen already from the analysis of description of 2-opt presented in Section 2.3, the task that can be done in parallel is the exploration of neighbourhood of the current solution. This is divided between individual CUDA blocks. Each block explores approximately the same amount of possible neighbours to the current solution (in the worst case, when no improving solution is found), including the cost evaluation. However, if it finds an improving solution, that solution is stored into the global memory allocated for each block, and the block terminates. If at least one of the blocks found an improving

solution, the minimal cost solution amongst all blocks is found and stored into memory as the current solution for the next iteration. Otherwise, the current solution is returned. The cost function evaluation itself was not parallelized, in each block only a single thread performs this task.

The outline of the parallel algorithm can be given as follows:

Step 1: Set current solution S = Initial solution

Step 2: Explore the neighbourhood of S by G blocks in parallel. In each block b:

- **Step 1.1:** Determine initial index i for b
- **Step 1.2:** Explore all neighbours of S created by swapping of i and $j, j \in \{1, \ldots, N\}$. If improving neighbour T found, go to *step 1.4*.
- **Step 1.3:** Determine next index i for b. If $i \geq N$, terminate. Otherwise go to *step 1.2*.
- **Step 1.4:** Store T and its objective function value f_T into global memory and terminate.

Step 3: If no improving solution found, exit procedure and return S as the best solution found. Otherwise determine the best solution amongst those found by blocks in parallel.

Step 4: Store best solution as S. Go to *step 2*

Where N is the number of jobs in the schedule and i is the outer loop index (see pseudocode 4 for sequential version of 2-opt).

3.2.1 Exploration and Evaluation of neighbouring solutions

In this kernel, the neighbours of the solution are generated and evaluated. It is obvious from the sequential version pseudocode (Algorithm 4), that the tasks of generating individual neighbours by swapping every possible pair of jobs (i, j) for $i = 1, \ldots, N$ and $j = i + 1, \ldots, N$ can be considered independent and executed in parallel. The shared memory is used to store the neighbouring solution which is being created and evaluated, as well as 2 rows of completion times matrix necessary for the flowshop makespan evaluation. If the new solution is better than the current one, it is stored into global memory allocated for each block, to avoid data races between blocks (this is illustrated in Figure 14 depicting memory layout for six jobs and four blocks). The improvements counter in global memory is incremented using atomic operation, to reflect this. This counter is compared against zero after the kernel termination, to determine if the stopping criterion of the algorithm was reached. The cost function itself is evaluated by a single thread only; the other threads of a block process the elements of the solution when transferring data between memory locations.

It is however impractical to allocate full number of $(N-1)^2/2$ blocks on the GPU for the most cases, as this number can be very large, whereas the number of SMs and

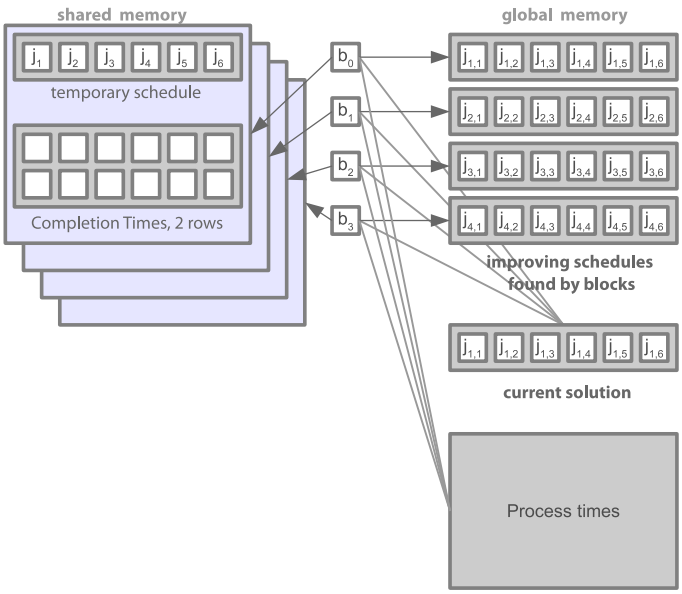

Figure 14: CUDA based 2-opt memory layout

the number of resident blocks on SM is limited by various factors (discussed in Section 2.8), such as the number of threads in a block and a registers/shared memory usage. The roughly optimal number of threads in a block maximizing the number of resident blocks, as well as GPU occupancy, is therefore determined based on the calculations performed in the CUDA occupancy calculator tool [79], as a function of the number of jobs in a schedule (which determines the size of shared memory used). This maximizes the utilization of the GPU, reduces the total global memory size required by grid, as well as the workload done by the search for minimal cost solution in the next kernel, however the mapping of the blocks to the tasks becomes more complicated.

Under the assumption that the number of blocks will be nearly always smaller than the aforementioned function of the number of actual jobs for the problem instances of interest (problems with schedules longer than 30), only the outer loop of the sequential 2-opt algorithm was parallelized. The inner loop is performed by each block sequentially. This reduces the data transfers between global and shared memory, and doesn't eliminate the advantage of the low complexity of the swap operation at the same time. If the solution created by swapping jobs i and j is worse than the current one, it is easy to reverse this change by swapping again j and i, with constant complexity. Therefore maximally $N - 1$ blocks are needed. The mapping of blocks to tasks is illustrated in Figure 15.

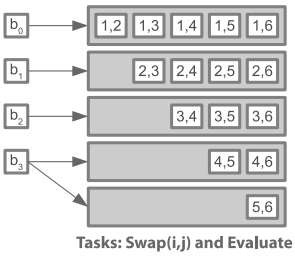

Figure 15: CUDA based 2-opt, mapping of blocks to tasks

3.2.2 Finding minimal cost index in parallel

This procedure is described in detail in the NEH implementation description, 3.1. The only difference here is, that the smaller number of blocks was used, therefore the searched array size is given by the size of the grid. The best solution's index and cost are stored for the later call to the kernel, which copies the solution at index into the current best solution memory, also described in the NEH Section.

3.3 Chaos driven DABC

All of the original variants of DABC, as well as the ABC algorithms (described in Sections 2.5 and 2.4), pay very little attention to stochasticity, despite making heavy use of randomness in their workflow. The stock DABC doesn't suggest any particular pseudo random number generator, and presumes the usage of one of the standard ones. The most commonly used PRNG in evolutionary algorithms is Mersenne Twister, owing to its good stochastic properties and speed [70]. This thesis presents the possibility to improve the standard DABC algorithm by exchanging the PRNG for a chaos based PRNG.

In the first part of research, nine unique chaotic systems have been included as CPRNG for the DABC: Arnold's cat map, Burgers map, Delayed Logistic, Dissipative Standard map, Henon, Ikeda, Lozi, Sinai and Tinkerbell maps. These new chaos embedded algorithms (hereafter referred to as variants) can be collectively labelled as **CDABC**. The basic premise of this work was to ascertain if any improvement can be achieved in DABC by using chaotic systems in place of PRNG. The experimental results for this with application to FSSLS with setup time and the FSSNW problem are presented in Sections 4.3 and 4.4.

The second part of research concentrates on the CPRNGs that have proven to be the best performing ones, i.e. four of the original nine CDABC variants: $\mathrm{CDABC_L}$ using Lozi map, $\mathrm{CDABC_{DL}}$ using Delayed Logistic map, $\mathrm{CDABC_B}$ with Burgers map and $\mathrm{CDABC_T}$ with Tinkerbell map. The experimentation for these algorithms applied to solving QAP and CVRP problems is described in Section 4.5.

3.3.1 Chaotic pseudo random number generator

There are two options how to utilize the chaos maps to implement the CPRNG. Either the large sequence of numbers – the inception of chaos map – is stored in a file, and used iteratively. This approach, however, consumes large amount of memory. The second option is to use random starting point on the map, and a mathematical equation to get the next points. This concept is similar to the PRNG seed. In this way, it is no longer necessary to store and read large data. For CDABC variants, the latter approach was used. The pseudocode describing this concept is given in Algorithm 5. Values on X-axis are used as the output sequence, whereas values on Y-axis are merely internal memory of the map.

The DABC algorithm structure remains almost the same (as described in pseudocode 3), the only alteration lies in replacement of calls to get next pseudo-random number from PRNG for calls to CPRNG. The detailed CDABC pseudocode is presented in Algorithms 6 to 16, emphasising the lines which lead to the calls to the pseudo-random number generator that were replaced by the chaos-based one by the red color. These parts are further described in the text and the analysis of the quantity of random numbers generated is provided.

1 **begin** CPRNG_init:
  Input: $seed$
  /* set initial point on map: */
2 $X \leftarrow seed$ (mod 0.1)
3 $Y \leftarrow X$
4 **end**
5 **begin** CPRNG_get_next:
  /* compute next X and Y according to given chaotic map equation: */
6 $X^{(n+1)} \leftarrow f_x(X^{(n)}, Y^{(n)}, Y^{(n+1)})$
7 $Y^{(n+1)} \leftarrow f_y(X^{(n)}, Y^{(n)}, X^{(n+1)})$
8 **return** $X^{(n+1)}$ *(mod 1)*
9 **end**

Algorithm 5: CPRNG pseudocode

3.3.2 DABC and CDABC randomness

It has been stated before that DABC heavily relies on stochasticity. Here, the verification of this claim is attempted at, and the quantification of the random numbers (RN) required during the course of algorithm flow is specified. Starting from the pseudocode 6, it can be observed that the random number generation is involved during the main iteration in employed bee, onlooker bee, as well as scout bee phase (lines 11-13), as well as in the initialization phase in population and the list of operations initialization (lines 3 and 8). The list of operations NL is part of the implementation of adaptive strategy described in Section 2.5.

To initialize NL, constant number of NL_L RN is needed (where NL_L is length of NL, specified as parameter of DABC), as can be seen in Algorithm 16. To initialize the population according to Algorithm 13, at least $FS \times n$ RN numbers are needed, where n is number of jobs in a schedule. However, in reality, this number will be certainly much higher. An experiment has been conducted to obtain the average number of RN need in this algorithm with Mersenne Twister for 30 solutions and number of jobs from 10 to 50. The excerpt from results is shown in table 3, where only every 10-th row was selected. Mean gives the average quantity of random numbers needed for the initialization of all solutions in the population, Mean/Element column gives the average number of RN needed for an element in the population (Mean / ($FS \cdot n$), where FS is number of food sources and n number of jobs). With the correlation coefficient of 0.976, there is linear dependency of RN needed on the number of jobs, which can be formulated, using the results of linear regression, as follows: $RNE = 0.037 \cdot n + 2.804$, where n is number of jobs and RNE is the average number of RN/element of population. In total in the initialization phase, average $RN = FS \cdot n \cdot RNE + NL_L = FS \cdot n \cdot (0.037 \cdot n + 2.804) + NL_L$.

In the employed bee phase, presented in Algorithm 7, for each food source, RN needed are between 2 and $4 + 1 + (Loop_{max} \cdot 2) + (NL_L - WNL_L)$. The minimal quantity of RN is needed when the operation is simple insert or simple swap (12), and the

Input:
FS : number of food sources
$Limit$: limit of improvements trial at a food source
$Loop_{max}$: maximal number of iterations in local search
PL : probability of local search
T : number of iterations of algorithm
NL_L : length of NL list
WNL_L : length of WNL list

```
1  begin initialize:
       /* read in problem data                                          */
2      problem ← read_problem()
       /* generate FS food sources as random permutations of sequence 1 to
          number_of_jobs(problem)                                       */
3      P.π ← random_init( FS, 1, number_of_jobs(problem))
       /* evaluate food sources                                         */
4      P.fitness ← evaluate( P.π )
       /* initialize limits                                             */
5      P.limit ← array(FS)
       /* initialize WNL and NL, randomly fill NL                       */
6      WNL ← array(WNL_L)
7      NL ← array(NL_L)
8      refill(NL,WNL)
9  end
10 for t = 1 to T do
       /* 1.send employed bees to food sources:                         */
11     employed_bee( P, PL, Loop_max, NL, WNL )
       /* 2.let onlooker bees select food sources:                      */
12     onlooker_bee( P, NL, WNL )
       /* 3.send the scout to search new food source:                   */
13     scout_bee( P,Limit )
       /* 4.memorize best food source:                                  */
14     memorize_best( P )
15 end
16 begin memorize_best
       Data: population P
       Result: modified population P
17     P.π_best ← P.π_i, where: ∀k, k ≠ i : P.fitness_i ≤ P.fitness_k
18     P.fitness_best ← P.fitness_i
19 end
```

Algorithm 6: CDABC pseudocode

```
1  begin employed_bee
       Data: population P, PL, Loop_max, NL, WNL
       Result: modified population P
       /* for each food source:                                          */
2      foreach π_i in P.π do
           /* get operation from Adaptive strategy list:                 */
3          operation ← get_operation(NL,WNL)
           /* perform operation on π_i:                                  */
4          π_i^(c) ← operation(π_i)
5          fitness_i^(c) ← evaluate( π_i^(c) )
6          if fitness_i^(c) ≤ P.fitness_i then
               /* local search on π_i^(c) with probability PL            */
7              if CPRNG_get_next() < PL then
8                  π_i^(c), fitness_i^(c) ← local_search( π_i^(c), fitness_i^(c), Loop_max, last_operation(NL) )
9              end
10             P.π_i ← π_i^(c)
11             P.fitness_i ← fitness_i^(c)
12             P.limit_i ← 0
               /* update adaptive strategy WNL                           */
13             update_WNL(WNL,operation)
14         else
15             P.limit_i ← P.limit_i + 1
16         end
17     end
18 end
```

Algorithm 7: CDABC pseudocode, employed bee

```
1  begin onlooker_bee
       Data: population P, NL, WNL
       Result: modified population P
       /* for each onlooker bee (using FS onlooker bees):                    */
2      for i = 1 to FS do
           /* randomly pick two food sources π_{r1}, π_{r2}                  */
3          r1, r2 ← get_unique_rand2( 1, FS )
           /* select better of π_{r1}, π_{r2}                                */
4          if P.fitness_{r1} ≤ P.fitness_{r2} then
5              s ← r1
6          else
7              s ← r2
8          end
           /* get operation from Adaptive strategy list                      */
9          operation ← get_operation(NL,WNL)
           /* perform operation on π_s:                                      */
10         π_s^(c) ← operation(P.π_s)
11         fitness_s^(c) ← evaluate( π_s^(c) )
12         if fitness_s^(c) ≤ P.fitness_s then
13             P.π_s ← π_s^(c)
14             P.fitness_s ← fitness_s^(c)
15             P.limit_s ← 0
               /* update adaptive strategy WNL                               */
16             update_WNL(WNL,operation)
17         else
18             P.limit_s ← limit_s + 1
19         end
20     end
21 end
```

Algorithm 8: CDABC pseudocode, onlooker bee

```
1  begin scout_bee
       Data: population P, Limit
       Result: modified population P
       /* select a solution which was not successfully improved given number
          of trials:                                                         */
2      s ← i, where: P.limit_i ≥ Limit
3      if s ∈ [1, FS] then // if such solution exists
4
           /* perform 3 insert operations on P.π_best                        */
5          P.π_s ← insert( P.π_best )
6          P.π_s ← insert2( P.π_s )
7          P.fitness_s ← evaluate( P.π_s )
8          P.limit_s ← 0
9      end
10 end
```

Algorithm 9: CDABC pseudocode, scout bee

```
   Input: π, fitness, Loop_max, last_operation
1  begin select_operation:
2      if last_operation ∈ {insert, insert2} then
3          operation ← swap
4      else
5          operation ← insert
6      end
7  end
8  begin
9      π^(a) ← π
10     fitness^(a) ← fitness
11     for i = 1 to Loop_max do
12         π^(c) ← operation(π^(a))
13         fitness^(c) ← evaluate( π^(c) )
14         if fitness^(c) ≤ fitness^(a) then
15             π^(a) ← π^(c)
16             fitness^(a) ← fitness^(c)
17         end
18     end
19     return π^(a), fitness^(a)
20 end
```

Algorithm 10: CDABC pseudocode, local search

```
1  begin get_unique_rand2:
       Input: L, U
       /* get 2 unique random integers from range [L, U]                    */
2      r1 ← value_to_range( CPRNG_get_next() * MAX_INT, L, U )
3      r2 ← value_to_range( CPRNG_get_next() * MAX_INT, L, U )
4      if r2 = r1 then
           /* advance r2 to the next position in [L, U]                     */
5          r2 ← next_in_range(r2,L,U)
6      end
7      return r1, r2
8  end
9  begin value_to_range:
       Input: v, L, U
10     return v (mod (U − L + 1)) + L
11 end
12 begin next_in_range:
       Input: v, L, U
13     return (v − L + 1) (mod (U − L + 1)) + L
14 end
```

Algorithm 11: CDABC get two unique random integers in range $[L, U]$

```
1  begin swap:
       Input: solution
       /* select 2 random non-identical positions in the schedule:          */
2      r_1, r_2 ← get_unique_rand2( 1, length(Solution) )
       /* swap jobs in the solution                                         */
3      solution_n ← swap_jobs(solution, r_1, r_2)
4      return solution_n
5  end
6  begin insert:
       Input: solution
       /* select 2 random non-identical positions in the schedule:          */
7      r_1, r_2 ← get_unique_rand2( 1,length(Solution) )
       /* insert                                                            */
8      if r_1 < r_2 then
9          solution_n ← left_shift_jobs(solution, r_1, r_2)
10     else
11         solution_n ← right_shift_jobs(solution, r_1, r_2)
12     end
13     return solution_n
14 end
15 begin insert2:
       Input: solution
16     solution_n ← insert( solution )
17     solution_n ← insert( solution_n )
18     return solution_n
19 end
20 begin swap2:
       Input: solution
21     solution_n ← swap( solution )
22     solution_n ← swap( solution_n )
23     return solution_n
24 end
```

Algorithm 12: CDABC Operation

```
1  begin random_init:
       Input: FS,L,U
2      π ← []
3      for i = 1 to FS do
4          π_i ← []
5          for j = 1 to U − L + 1 do
6              while π_{i,j} not set do
                   /* get random job from range [L, U]                    */
7                  π_{i,j} ← value_to_range( CPRNG_get_next() * MAX_INT, L, U )
                   /* verify if job not present in π_i                    */
8                  for t = 1 to j − 1 do
9                      if π_{i,t} = π_{i,j} then
10                         unset( π_{i,j} )
11                         break
12                     end
13                 end
14             end
15         end
16     end
17     return π
18 end
```

Algorithm 13: CDABC pseudocode, random initialisation

```
1  begin get_operation :
       Data: list of operations NL, list of successful operations WNL
       Result: modified NL, selected operation
2      if is_empty(NL) then
3          refill(NL,WNL)
4      end
5      operation ← pop(NL)
6      return operation
7  end
```

Algorithm 14: CDABC get operation from list of operations NL

```
1  begin update_WNL :
       Data: list of successful operations WNL, operation
       Result: modified WNL
       /* insert operation into WNL at next position                      */
2      i ← top_pointer(WNL) mod length(WNL) + 1
3      WNL_i ← operation
4  end
```

Algorithm 15: CDABC update WNL

```
1  begin refill:
       Data: list of operations NL, list of successful operations WNL
       Result: modified NL and WNL
2      Operations ← [insert, insert2, swap, swap2]
3      if not empty(WNL) then
4          for i=1 to length( NL ) do
5              if i ≤ length(WNL) then
                   /* Fill first length(WNL) operations using all operations
                      present in WNL                                          */
6                  w ← i mod top_pointer(WNL) + 1
7                  NL_i ← WNL_w
8              else
                   /* Fill the rest using randomly selected operations from 4
                      available                                               */
9                  o ← value_to_range( CPRNG_get_next() * MAX_INT, 1, 4)
10                 NL_i ← Operations_o
11             end
12         end
           /* empty WNL                                                      */
13         clear(WNL)
14     else
           /* Fill entirely using randomly selected operations               */
15         for i=1 to length( NL ) do
16             o ← value_to_range( CPRNG_get_next() * MAX_INT, 1, 4)
17             NL_i ← Operations_o
18         end
19     end
20 end
```

Algorithm 16: CDABC refill list of operations NL

new solution found doesn't improve upon the previous one. Insert and swap require both only 2 random numbers. However, the operation used for the generation of the new solution can be one of the composed operations: insert2, swap2. In such case, 4 random numbers are needed. If the solution does improve upon its predecessor, another RN is needed to determine if the local search should happen. Inside the local search, only simple insert and simple swap operations are needed, which happen in every local search iteration, fixed number of times, $Loop_{max}$, hence $Loop_{max} \cdot 2$ random numbers are needed inside local search. If the NL list is empty and has to be refilled, another $(NL_L - WNL_L)$ random numbers are needed (where NL_L is length of NL list, WNL_L length of WNL list). Therefore in employed bee phase, RN is in the range from $FS \cdot 2$ to $FS \cdot (5 + (2Loop_{max}) + (NL_L - WNL_L))$.

In the onlooker bee phase, shown in Algorithm 8, the number of RNs is between $2 + 2 = 4$ and $2 + 4 + (NL_L - WNL_L)$ for a solution. Explanation is similar as in the employed bee phase. Two randomly selected solutions are always needed. On the better one, an operation is performed, requiring either two or four randomly selected jobs positions. This makes up the total of $FS \cdot 4 - FS \cdot 6 + (NL_L - WNL_L)$ random numbers.

The scout bee phase, presented in Algorithm 9, is the simplest, requiring between 0 and 6 random numbers in total. If no solution crosses the $Limit$, no operation is performed. If a solution with exceeded limit exists, exactly 6 random numbers are needed to perform 3 insert operations.

In the main iteration, number of random numbers used is between $FS \cdot 6$ and $FS \cdot (2Loop_{max} + 2(NL_L - WNL_L) + 17)$ per iteration, $T \cdot FS \cdot 6$ and $T \cdot FS \cdot (2Loop_{max} + 2(NL_L - WNL_L) + 17)$ in total. Total quantity of random numbers needed in the algorithm is therefore between $RN_{init} + T \cdot RN_{iter} = (FS \cdot n \cdot (0.037 \cdot n + 2.804) + NL_L) + T \cdot (FS \cdot 6)$ and $RN_{init} + T \cdot RN_{iter} = (FS \cdot n \cdot (0.037 \cdot n + 2.804) + NL_L) + T \cdot FS \cdot (2Loop_{max} + 2(NL_L - WNL_L) + 17)$. As an example, with the default parameter settings described in 2.5, where $FS = 30, Loop_{max} = 200, T = 100, NL_L = 20$ and $WNL_L = 15$, for the number of jobs $n = 100$, this gives the lower bound of $37.5 \cdot 10^3$, and the upper bound of $1.3 \cdot 10^6$ random numbers. This analysis gives only the coarse lower, respective upper bound of the quantity of random numbers needed, as the dependencies are neglected for the sake of simplification. Nevertheless, it can be seen that these values are very large, and the use of stochasticity in the algorithm is intensive.

Table 3: Random numbers required in random initialization of population

Jobs	Elements	Mean	Std	Mean/Element
10	300	878.200	53.156	2.927
15	450	1,515.900	82.313	3.369
20	600	2,142.100	114.319	3.570
25	750	2,895.867	181.762	3.861
30	900	3,607.433	218.697	4.008
35	1050	4,309.767	147.940	4.105
40	1200	5,144.633	279.140	4.287
45	1350	5,859.933	220.570	4.341
50	1500	6,786.667	306.522	4.524

3.4 Centralities Based ABC

In the Adaptive ABC algorithm, the complex network analysis was used for adaptive control of the population. The structure of the algorithm is as follows: firstly, the weighted adjacency matrix is being created throughout the algorithm iterations, for some fixed number of iterations, a fraction of the total expected number of iterations before algorithm termination. The complex network recorded this way is then analysed, and this information is subsequently used to identify the nodes (solutions) that don't play a significant role in the population dynamics. In this algorithm, such nodes are replaced by the new randomly generated ones, although different schemas of the replacements generation could also be used.

The measures used to identify the nodes that do not contribute significantly to the population improvement were chosen to be the three types of vertex centrality, the weighted degree centrality (strength), closeness and betweenness centrality, as described in section 2.7.2.

The vertices representing solutions are ranked according to these measures, and the fixed ratio of the solutions corresponding to the lowest ranking nodes is removed and regenerated. The adjacency matrix is then reset. The entire procedure of network recording and the nodes ranking and replacement is repeated until the algorithm terminates. This concept is illustrated in figures 16 - 19.

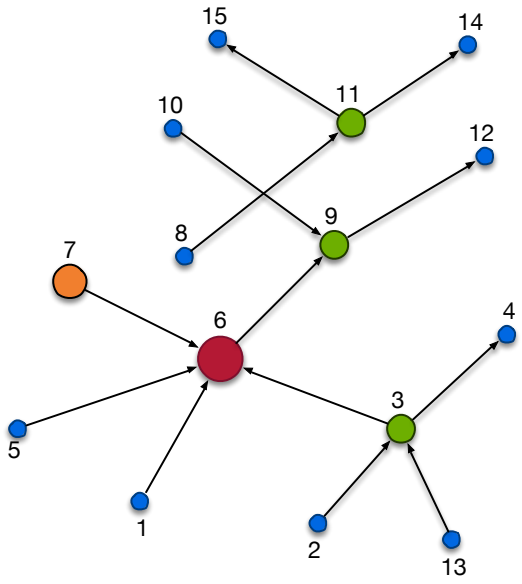

Figure 16: The network with labelled nodes ranked by centrality. The larger centrality nodes are marked in bigger size and different colors. The smallest blue nodes have the lowest centrality, the largest red node has the highest centrality value.

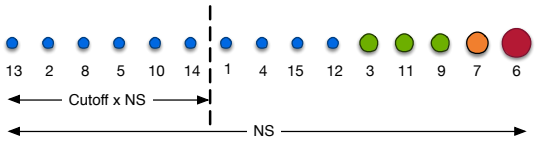

Figure 17: The nodes sorted according to their centrality score in ascending order. The first *Cutoff* $\times$ *NS* nodes will be removed.

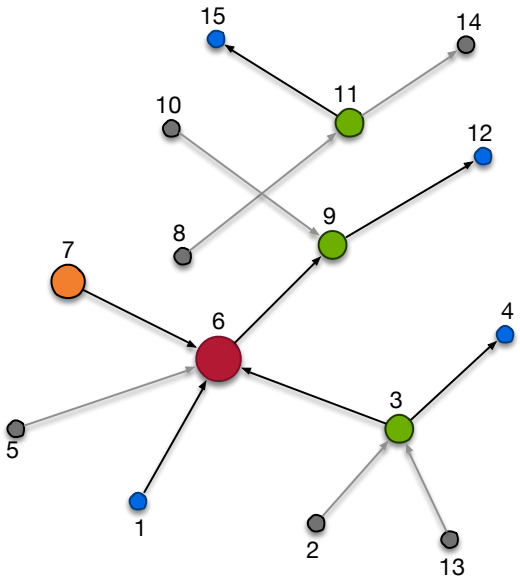

Figure 18: The nodes marked in gray will be removed from the network.

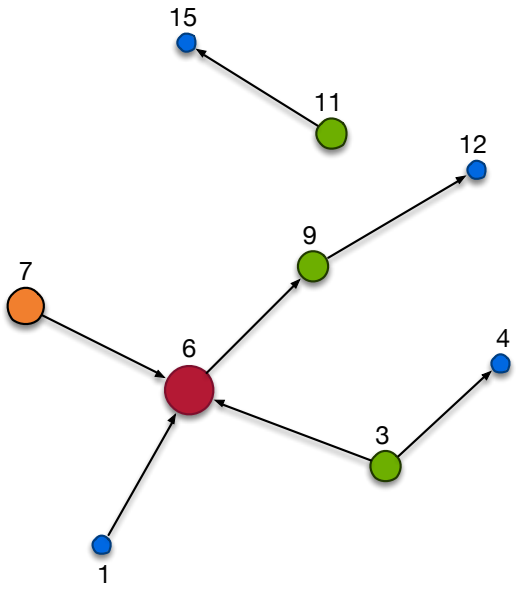

Figure 19: The network after the low centrality nodes removal. The most important nodes are preserved.

Employing these three distinct vertex influence measures, two variants of the Adaptive ABC algorithm were created, both combining the usage of all three of them: the 1st variant (Adaptive ABC 1, Algorithm 17) in the phase of network analysis conceptually splits the population into three parts. Each of them is evaluated using different centrality measure type on the same network. As described in the pseudocode, in the network analysis and nodes pruning phase of the algorithm, first $\frac{NS}{3}$ nodes are sorted, according to the first centrality measure (weighted degree centrality). $\frac{NS}{3} \times Cutoff$ lowest ranking nodes are recreated, and the same procedure is repeated for the second third of the population using closeness, and for the last third of the population using betweenness.

The 2nd variant (Adaptive ABC 2, Algorithm 18) uses three fully separated sub-populations, each with their own network. Each of them is evaluated using different centrality measure type. After the less influential nodes are pruned and the networks reset, the tournament selection of size two is performed to select every next-generation solution of every sub-population, choosing the better of two solutions randomly selected from all three sub-populations. In this way the information sharing between sub-populations is ensured.

In order to explore the influence of distinct centralities on the speed of convergence, a simpler version of Adaptive ABC was created. This version, named Adaptive ABC 3, takes the centrality measure to be used for network analysis and nodes evaluation as a parameter. The algorithm is described by pseudocode given in 19.

All of the previously discussed Adaptive ABC variants determine the nodes to be

Input:
NS : number of solutions in the population
NGenNumber : number of generations for network creation
CutoffRatio : ratio of low centrality ranking nodes to be recreated

```
1
2  Generate initial population Population of NS solutions
3  Initialise the network Network
4  initialise generation counter GenCounter
5  initialise network generation counter NGenCounter
6  SubPopulationSize ← 1/3 × NS
7  while max generation not reached do
8      send Employed bees and update the Network
9      send Onlooker bees and update the Network
10     send Scout bees
11     memorize the best solution
12     if NGenCounter == NGenNumber then
13         for i=1 to 3 do
14             for nodes in range i-1 × SubPopulationSize + 1 to i × SubPopulationSize do
15                 calculate centrality i of nodes
16                 sort nodes by the centrality ranking in ascending order
17                 replace the solutions belonging to the first SubPopulationSize ×
                   CutoffRatio nodes by new ones
18             end
19         end
20         reset Network and NGenCounter
21     end
22     increase GenCounter
23     increase NGenCounter
24 end
```

Algorithm 17: Adaptive ABC 1

Input:
NS : number of solutions in the population
NGenNumber : number of generations for network creation
CutoffRatio : ratio of low centrality ranking nodes to be recreated

```
1
2  Generate 3 initial sub-populations SubPopulation(1), SubPopulation(2), SubPopulation(3) of
   NS/3 solutions
3  foreach SubPopulation(i) do
4  |  Initialise the Network(i)
5  end
6  initialise generation counter GenCounter
7  initialise network generation counter NGenCounter
8  while max generation not reached do
9  |  foreach SubPopulation(i) do
10 |  |  send Employed bees and update the Network(i)
11 |  |  send Onlooker bees and update the Network(i)
12 |  |  send Scout bees
13 |  |  memorize the best solution
14 |  end
15 |  memorize the subpopulation with the best solution
16 |  if NGenCounter == NGenNumber then
17 |  |  foreach SubPopulation(i) do
18 |  |  |  calculate centrality of all NS nodes
19 |  |  |  sort nodes by the centrality ranking in ascending order
20 |  |  |  replace the solutions belonging to the first NS × CutoffRatio nodes by new ones
21 |  |  |  reset Network(i)
22 |  |  end
23 |  |  Population ← join(SubPopulation(1), Subpopulation(2), Subpopulation(3))
24 |  |  foreach SubPopulation(i) do
25 |  |  |  foreach Solution(s), s ∈ {1, .., NS/3} do
26 |  |  |  |  Tournament selection size 2 from Population
27 |  |  |  end
28 |  |  end
29 |  |  reset NGenCounter
30 |  end
31 |  increase GenCounter
32 |  increase NGenCounter
33 end
```

Algorithm 18: Adaptive ABC 2

Input:
NS : number of solutions in the population
NGenNumber : number of generations for network creation
CutoffRatio : ratio of low centrality ranking nodes to be recreated
MeasureType : centrality measure type to be used

```
1
2  Generate initial population Population of NS solutions
3  Initialise the network Network
4  initialise generation counter GenCounter
5  initialise network generation counter NGenCounter
6  while max generation not reached do
7  |  send Employed bees and update the Network
8  |  send Onlooker bees and update the Network
9  |  send Scout bees
10 |  memorize the best solution
11 |  if NGenCounter == NGenNumber then
12 |  |  calculate centrality MeasureType of all NS nodes
13 |  |  sort nodes by the centrality ranking in ascending order
14 |  |  replace the solutions belonging to the first NS × CutoffRatio nodes by new
   |  |  ones
15 |  |  reset Network and NGenCounter
16 |  end
17 |  increase GenCounter
18 |  increase NGenCounter
19 end
```

Algorithm 19: Adaptive ABC 3

removed from the population merely by their vertex centrality ranking, in accordance with the main premise of this research. The quality of the solutions themselves is not taken into consideration. This, however, can lead to the removal of the good quality solutions from the population, in case they haven't contributed to the other solutions for given number of generations, therefore somewhat more randomizing the search. The convergence could be faster if some of the good solutions were maintained, i.e. if the quality of the solution was also considered when determining which nodes to remove. From this idea stem two more variants, based on the Adaptive ABC 3: Adaptive ABC 3.b, described in 20, and the Adaptive ABC 3.c, described in 21.

Both algorithms work in the similar fashion: two sorted lists of nodes are created. One of them sorts nodes by the centrality ranking, using the centrality measure type provided as parameter. The other one sorts nodes by the solution costs or fitness, so that the nodes belonging to the solutions with the best fitness come first. Certain number of solutions with the best fitness values is always preserved, regardless of the centrality ranking of the respective vertices in the network. This number is determined by the parameter *EliteRatio*, giving the ratio of the best solutions from entire population to always maintain during the nodes pruning.

In case of the Adaptive ABC 3.b, the first $Cutoff \times NS$ nodes are considered for the removal in the same way as in the previously described algorithm variants. However, if any of these nodes belongs to the first $EliteRatio \times NS$ best solutions, it is skipped in the pruning process. Hence this algorithm recreates $Cutoff \times NS$ nodes or less in the network analysis and nodes pruning phase.

The Adaptive ABC 3.c, on the other hand, in the most cases recreates the full number of $Cutoff \times NS$ nodes. It goes through the entire list of the nodes sorted by centrality ranking. The removed solutions are counted during the procedure of the nodes pruning. If the low ranking node is amongst the first $EliteRatio \times NS$ best solutions, it is again skipped. In the opposite case, it is replaced by the new solution, and the removed solutions counter is increased. The procedure does not end before the counter reaches value of $Cutoff \times NS$, or the end of the nodes list is reached.

Input:
NS : number of solutions in the population
NGenNumber : number of generations for network creation
CutoffRatio : ratio of low centrality ranking nodes to be recreated
MeasureType : centrality measure type to be used *EliteRatio* : ratio of the best cost solutions to always keep in the population

```
1
2  Generate initial population Population of NS solutions
3  Initialise the network Network
4  initialise generation counter GenCounter
5  initialise network generation counter NGenCounter
6  while max generation not reached do
7  |  send Employed bees and update the Network
8  |  send Onlooker bees and update the Network
9  |  send Scout bees
10 |  memorize the best solution
11 |  if NGenCounter == NGenNumber then
12 |  |  calculate centrality MeasureType of all NS nodes
13 |  |  nodesByCentrality ← sort nodes by the centrality ranking in ascending order
14 |  |  nodesByCost ← sort nodes by corresponding solutions' fitness value in
   |  |  ascending order
15 |  |  for i=1 to NS × CutoffRatio do
16 |  |  |  node ← nodesByCentrality[i]
17 |  |  |  if node not in nodesByCost[1 to NS × EliteRatio] then
18 |  |  |  |  replace the solution belonging to the node by new one
19 |  |  |  end
   |  |  |  // else skip node
20 |  |  end
21 |  |  reset Network and NGenCounter
22 |  end
23 |  increase GenCounter
24 |  increase NGenCounter
25 end
```

Algorithm 20: Adaptive ABC 3.b

Input:
NS : number of solutions in the population
NGenNumber : number of generations for network creation
CutoffRatio : ratio of low centrality ranking nodes to be recreated
MeasureType : centrality measure type to be used *EliteRatio* : ratio of the best cost solutions to always keep in the population

```
1
2  Generate initial population Population of NS solutions
3  Initialise the network Network
4  initialise generation counter GenCounter
5  initialise network generation counter NGenCounter
6  while max generation not reached do
7      send Employed bees and update the Network
8      send Onlooker bees and update the Network
9      send Scout bees
10     memorize the best solution
11     if NGenCounter == NGenNumber then
12         calculate centrality MeasureType of all NS nodes
13         nodesByCentrality ← sort nodes by the centrality ranking in ascending order
14         nodesByCost ← sort nodes by corresponding solutions' fitness value in
           ascending order
15         removed ← 0
16         for i=1 to NS do
17             node ← nodesByCentrality[i]
18             if node not in nodesByCost[1 to NS × EliteRatio] then
19                 replace the solution belonging to the node by new one
20                 removed ← removed + 1
21             end
22             if removed ≥ NS × CutoffRatio then
23                 break
24             end
25         end
26         reset Network and NGenCounter
27     end
28     increase GenCounter
29     increase NGenCounter
30 end
```

Algorithm 21: Adaptive ABC 3.c

4 Experimentation

The following text describes the experimentation performed for the implemented algorithms solving selected optimisation problems. It provides the information on the problem instances solved, as well as parameter setting and the hardware used for testing. First five parts, the NEH, 2-opt algorithm, CDABC for FSSLS, FSSNW, QAP and CVRP belong to the combinatorial optimisation domain, the last part, Centralities Based ABC, solves the standard continuous optimisation test functions. Experiment outputs are provided on CD (see appendix, A). Statistical analysis of this data of each experiment is found in Section 5.

4.1 NEH

This section presents the experimentation performed on the standard and CUDA accelerated implementation of NEH heuristic, comparing execution times. NEH is used to solve the problem of minimising the makespan of permutative flowshop scheduling problem. In order to obtain the input data, which would be large enough, Taillard Data Sets were extended with the new larger problem instances. The experimental data are described in the following text parts. The analysis of results is discussed in Section 5.1.

4.1.1 Extended Taillard Data Sets

The entire experimentation is conducted on the Taillard flowshop data sets. The original data sets comprise of twelve data sizes (indexed hereafter as (*jobs* x *machines*)); 20 x 5, 20 x 10, 20 x 20, 50 x 5, 50 x 10, 50 x 20, 100 x 5, 100 x 10, 100 x 20, 200 x 10, 200 x 20 and 500 x 20. The formulation of these data sets is given in [108].

One of the core premises of the GPU approach is the ability to set up massive parallel computation array to speed up the execution. Therefore, the original Taillard flowshop data sets (OTS) were extended to include new higher dimensional problems. Using the code snippet posted on the OR Library [111] by J. Beasley [9] detailing the Taillard code generation, four new higher dimensional data sets were generated, hereafter termed as the extended Taillard data sets (ETS). The sizes of the new data sets are 500 x 50, 700 x 20, 700 x 50 and 1000 x 20. Each data set contains ten unique instances. Using the Taillard code template, each new *seed* instance was uniquely generated within the range $\{2^{24}, (2^{31} - 1)\}$ using a uniform distribution. Using this seed, the new instance was generated. The full C code including all the new instances can be obtained from the repository at [22].

4.1.2 Experiment setup

The experimentation was conducted on the server housed at the Media Research Lab (MRL) at the VSB-Technical University of Ostrava. The specifications for the CUDA GPU are: Kepler generation, production line Tesla K20k, 4.8GB global memory, GPU Clock rate

of 706MHz with 2494 CUDA cores; CUDA Driver Version 5.5. The CPU specifications are: Intel Xeon CPU E5-2640 v2 with 2.00GHz processor and 8 CPU cores.

The experiments on all the afore mentioned problem instances of varying size (16 unique instance sizes, 10 instances of each size) were performed for both the CPU and the GPU based implementation. The resulting times in milliseconds were recorded and statistically analysed and compared. The excerpts from the results are given in the Tables 4 and 5 (both tables contain execution times and best schedules costs for first three problem instances of each size. Entire experimentation data were too large to fit the publication scope, but can be found on CD (appendix A). The statistical summary is given in the results analysis of NEH, in the afore mentioned Section 5.1.

Table 4: NEH on CPU, raw data excerpt.

Instance	Dimension	Time[ms]	Cost
ta001	5 x 20	0	725.85
ta002	5 x 20	0	782.5
ta003	5 x 20	0	705.45
ta011	10 x 20	0	1097.95
ta012	10 x 20	0	1179.6
ta013	10 x 20	0	1034.75
ta021	20 x 20	0	1711.4
ta022	20 x 20	0	1600.3
ta023	20 x 20	0	1721.8
ta031	5 x 50	10	1373.9
ta032	5 x 50	0	1468.84
ta033	5 x 50	0	1340.06
ta041	10 x 50	20	1838.26
ta042	10 x 50	0	1773.48
ta043	10 x 50	10	1691.98
ta051	20 x 50	20	2610
ta052	20 x 50	20	2473.04
ta053	20 x 50	20	2454.22
ta061	5 x 100	40	2683.96
ta062	5 x 100	40	2651.39
ta063	5 x 100	40	2547.08
ta071	10 x 100	80	3168.97
ta072	10 x 100	70	2914.06
ta073	10 x 100	90	3050.38
ta081	20 x 100	220	3856.11
ta082	20 x 100	170	3911.82
ta083	20 x 100	170	3886.27
ta091	10 x 200	640	5534.27
ta092	10 x 200	740	5459.82
ta093	10 x 200	810	5490955
ta101	20 x 200	1330	6380.72
ta102	20 x 200	1340	6506805
ta103	20 x 200	1340	6614165
ta111	20 x 500	19760	13904804
ta112	20 x 500	19540	14152622
ta113	20 x 500	19820	14018.59
ta121	50 x 500	51680	17313.26
ta122	50 x 500	52310	17456355
ta123	50 x 500	56570	17406168
ta131	20 x 700	52930	18998219
ta132	20 x 700	53180	18945346
ta133	20 x 700	53730	18938916
ta141	50 x 700	140330	22599326
ta142	50 x 700	141910	22717443
ta143	50 x 700	138580	22593352
ta151	20 x 1000	153180	26357402
ta152	20 x 1000	156180	26503836
ta153	20 x 1000	152210	26332857
ta161	50 x 1000	409729969	30324771
ta162	50 x 1000	403270031	30462154
ta163	50 x 1000	400140	30231104

Table 5: NEH on GPU, raw data excerpt.

Instance	Dimension	Time[ms]	Cost
ta001	5 x 20	1.08	725.85
ta002	5 x 20	1.074	782.5
ta003	5 x 20	1.069	705.45
ta011	10 x 20	1.644	1097.95
ta012	10 x 20	1.643	1179.6
ta013	10 x 20	1.637	1026.75
ta021	20 x 20	2.775	1711.4
ta022	20 x 20	2.783	1600.3
ta023	20 x 20	2.804	1721.8
ta031	5 x 50	4.798	1381.44
ta032	5 x 50	4.761	1468.84
ta033	5 x 50	4.774	1340.06
ta041	10 x 50	8.191	1838.92
ta042	10 x 50	8.23	1773.48
ta043	10 x 50	8.194	1691.98
ta051	20 x 50	15.118	2623.64
ta052	20 x 50	15.077	2473.04
ta053	20 x 50	15.085	2454.22
ta061	5 x 100	16.694	2683.96
ta062	5 x 100	16.686	2651.39
ta063	5 x 100	16.821	2547.08
ta071	10 x 100	30.171	3170.92
ta072	10 x 100	30.24	2914.06
ta073	10 x 100	30.215	3050.38
ta081	20 x 100	57.282	3853.99
ta082	20 x 100	57.326	3911.82
ta083	20 x 100	57.311	3886.27
ta091	10 x 200	112.925	5534.27
ta092	10 x 200	113.063	5459.82
ta093	10 x 200	112.84	5490.955
ta101	20 x 200	216.288	6380.72
ta102	20 x 200	215.998	6526.98
ta103	20 x 200	216.051	6614.165
ta111	20 x 500	4746.499	13833.76
ta112	20 x 500	4749.154	14152.622
ta113	20 x 500	4749.458	13991.408
ta121	50 x 500	11666.221	17313.26
ta122	50 x 500	11666.494	17456.355
ta123	50 x 500	11669.276	17501.094
ta131	20 x 700	19687.812	19113.73
ta132	20 x 700	19695.082	19011.922
ta133	20 x 700	19704.531	19038.971
ta141	50 x 700	48438.355	22660.654
ta142	50 x 700	48452.395	22834.645
ta143	50 x 700	48448.176	22681.174
ta151	20 x 1000	71390.695	26609.447
ta152	20 x 1000	71422.367	26785.055
ta153	20 x 1000	71299.961	26604.393
ta161	50 x 1000	175719.016	30651.863
ta162	50 x 1000	175845.5	30744.207
ta163	50 x 1000	175729.656	30450.488

4.2 2-opt algorithm

In this section, the experimentation performed on CUDA accelerated 2-opt and the sequential implementation of 2-opt for permutative flowshop with makespan criterion is described, comparing primarily the execution times, but also the solutions costs. The subset of standard Taillard data set was used (described in Section 4.4), with the problem instance size given in $machines \times jobs(m \times n)$ varying from 5×20 to 20×200. Eleven different sizes were used. Of each size, three different instances were evaluated. The description of the experiment setup follows, with the analysis of results given in 5.2.

4.2.1 Experiment setup

The experimentation was conducted on the server at the Media Research Lab (MRL) at the VSB-Technical University of Ostrava. The specification of the platform is given in 4.1.

The 2-opt algorithm requires an initial solution. In order to make simple and valid comparison between sequential and accelerated version, both were in each experiments started on the same solution defined by the jobs indices sorted in ascending order from 1 to n, where n is total number of jobs. The solution found slightly differs between sequential and parallel implementation, moreover, in parallel version the solution found is dependent on the number of blocks in the grid, however the semantics of the algorithm is similar. It could be expected that the accelerated version would produce better solutions on average, as the selection of the best solution from the solutions found by several blocks is incorporated in the algorithm. Because of this asymmetry, the analysis contains also the comparison of the solutions costs.

As stated before, the analysis of the experiments is presented in Section 5.2. The full experiment results are not included due to the space limitations, but can be found on CD (appendix A). The excerpt from the experiment results is presented in Tables 6 for the sequential version, and 7 for the parallel version. Only the solution costs and execution times are included.

Table 6: 2-opt algorithm on CPU, raw data excerpt.

Instance	Dimension	Time[ms]	Cost
ta001	20 x 5	10.000	719.650
ta002	20 x 5	20.000	773.750
ta003	20 x 5	0.000	686.100
ta011	20 x 10	10.000	1135.150
ta012	20 x 10	10.000	1188.600
ta013	20 x 10	0.000	1036.150
ta021	20 x 20	40.000	1732.700
ta022	20 x 20	40.000	1653.550
ta023	20 x 20	30.000	1710.400
ta031	50 x 5	1020.000	1337.020
ta032	50 x 5	720.000	1407.260
ta033	50 x 5	360.000	1331.560
ta041	50 x 10	840.000	1860.680
ta042	50 x 10	920.000	1773.200
ta043	50 x 10	1070.000	1690.080
ta051	50 x 20	1920.000	2644.400
ta052	50 x 20	1860.000	2493.760
ta053	50 x 20	1410.000	2500.960
ta061	100 x 5	20700.000	2599.780
ta062	100 x 5	19940.000	2517.350
ta063	100 x 5	26140.000	2442.560
ta071	100 x 10	27000.000	3118.270
ta072	100 x 10	31550.000	2859.560
ta073	100 x 10	31580.000	3003.790
ta081	100 x 20	36650.000	3888.720
ta082	100 x 20	27820.000	3948.160
ta083	100 x 20	32540.000	3973.800
ta091	200 x 10	736830.000	5419.690
ta092	200 x 10	887660.062	5374.070
ta093	200 x 10	852780.000	5391.405
ta101	200 x 20	1031000.000	6408.830
ta102	200 x 20	858579.938	6507.830
ta103	200 x 20	1028030.000	6571.595

Table 7: 2-opt algorithm on GPU, raw data excerpt.

Instance	Dimension	Time[ms]	Cost
ta001	20 x 5	23.215	727.400
ta002	20 x 5	21.931	779.000
ta003	20 x 5	25.332	693.800
ta011	20 x 10	22.705	1133.400
ta012	20 x 10	40.095	1182.500
ta013	20 x 10	37.502	1016.200
ta021	20 x 20	65.291	1738.300
ta022	20 x 20	69.687	1621.850
ta023	20 x 20	61.437	1771.850
ta031	50 x 5	666.402	1345.500
ta032	50 x 5	509.422	1413.320
ta033	50 x 5	363.188	1311.100
ta041	50 x 10	855.166	1826.540
ta042	50 x 10	903.844	1711.680
ta043	50 x 10	570.023	1697.600
ta051	50 x 20	1030.254	2685.180
ta052	50 x 20	1257.727	2563.140
ta053	50 x 20	1232.024	2459.820
ta061	100 x 5	4529.776	2615.760
ta062	100 x 5	5291.183	2537.040
ta063	100 x 5	5956.760	2447.880
ta071	100 x 10	6764.501	3104.000
ta072	100 x 10	4912.090	2897.560
ta073	100 x 10	5275.224	3091.010
ta081	100 x 20	9502.793	3862.780
ta082	100 x 20	7281.413	3943.160
ta083	100 x 20	9851.810	3869.120
ta091	200 x 10	61722.469	5392.215
ta092	200 x 10	59540.703	5404.975
ta093	200 x 10	61247.547	5432.610
ta101	200 x 20	62891.551	6459.505
ta102	200 x 20	68402.836	6522.060
ta103	200 x 20	72442.133	6592.815

4.3 Chaos driven DABC for FSSLS

The experiments performed on Chaos driven DABC for the lot-streaming flowshop scheduling problem with setup time, no-idle case, as well as chaos generated data sets used for testing, are described in the following text.

4.3.1 Chaotic maps generated data sets

In keeping with the theme of utilising the chaotic maps in lieu of PRNG, the data sets have been generated using two unique chaotic maps; the Lozi and the Dissipative map. Five unique sizes of data sets have been generated. They are from 10 jobs x 5 machines, 20 jobs x 10 machines, 50 jobs x 25 machines, 75 jobs x 30 machines and 100 jobs x 50 machines. There are 5 instances for each data set size, therefore, in total 25 data set instances for each of the Lozi and Dissipative data sets.

In order to have *unique* data sets, each *instance* was initialised from a unique *start* position of the respective chaotic system. Additionally, the map was not allowed to be reinitialised. Two different maps were used in order to gain more diversity, due to their different maps in the data sets, and to remove any particular bias when using any one system.

The datasets are available at [21] for download.

4.3.2 Experiment setup

The operating parameters of CDABC are given in Table 8. All parameters were kept constant for all the experimentation, in order not to introduce a bias. All experiments were conducted on the machine having Intel i7-3610QM CPU processor running at 2.3GHz with 8GB of RAM. All codes were written in the C programming language, compiled with the gcc 4.6.2.

Table 8: DABC Operating parameters, FSSLS

Parameter	Value
Food Source (FS)	30
Limit (food source)	50
$Loop_{max}$ (Local Search)	200
Local search probability (P_L)	0.2
Neighbourhood List (NL)	20
Winning Neighbourhood List (WNL)	0.75 x NL
Iterations	100

For each instance, fifteen (15) repeated experimentations were conducted in order to obtain statistical variance. Therefore, 375 individual experiments were conducted on the Lozi and Dissipative data sets, a total of 750 experimentations. The excerpts from the results for both data sets are given in Tables 9 and 10. Only cost values and execution times are shown for a selected subset of instances, for every instance and every CPRNG, only 3 experiments are shown out of actually conducted 15. The averaged results together

with analysis is given in Section 5.3. The full experiment results with schedules can be found on CD (appendix A).

Table 9: CDABC for FSSLS, Lozi data set, raw data excerpt.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
1	MT	0.195	541	24	MT	22.089	12312
1	MT	0.196	541	24	MT	26.279	12281
1	MT	0.185	541	24	MT	22.67	12373
1	AC	0.243	541	24	AC	19.793	12449
1	AC	0.236	541	24	AC	25.293	12544
1	AC	0.243	541	24	AC	21.909	12525
1	B	0.549	541	24	B	59.571	12231
1	B	0.559	541	24	B	59.94	12237
1	B	0.545	541	24	B	70.411	12231
1	DL	0.748	541	24	DL	87.717	12233
1	DL	0.754	541	24	DL	92.298	12190
1	DL	0.749	541	24	DL	77.252	12099
1	Dis	0.357	541	24	Dis	24.479	12430
1	Dis	0.419	541	24	Dis	25.735	12411
1	Dis	0.344	541	24	Dis	38.597	12351
1	Hen	0.232	541	24	Hen	21.482	12448
1	Hen	0.226	541	24	Hen	20.887	12603
1	Hen	0.218	541	24	Hen	21.238	12543
1	I	0.421	541	24	I	33.047	12393
1	I	0.424	541	24	I	33.887	12399
1	I	0.41	541	24	I	31.579	12337
1	Lozi	0.371	541	24	Lozi	43.571	12250
1	Lozi	0.385	541	24	Lozi	38.08	12273
1	Lozi	0.381	541	24	Lozi	39.243	12454
1	S	0.282	541	24	S	20.58	12513
1	S	0.269	541	24	S	21.266	12612
1	S	0.273	541	24	S	22.278	12464
1	T	0.697	541	24	T	72.793	12198
1	T	0.702	541	24	T	78.809	12193
1	T	1.166	541	24	T	75.34	12177
2	MT	0.19	430	25	MT	25.049	13103
2	MT	0.21	430	25	MT	22.368	13130
2	MT	0.205	430	25	MT	21.203	13038
2	AC	0.254	430	25	AC	22.902	13105
2	AC	0.254	430	25	AC	20.997	13237
2	AC	0.254	430	25	AC	22.293	13231
2	B	0.57	430	25	B	56.682	12922
2	B	0.549	430	25	B	56.498	12955
2	B	0.561	430	25	B	60.393	12959
2	DL	0.763	430	25	DL	89.408	13042
2	DL	1.557	430	25	DL	80.929	12851
2	DL	0.774	430	25	DL	83.357	13014
2	Dis	0.343	430	25	Dis	22.938	13193
2	Dis	0.341	430	25	Dis	32.031	13215
2	Dis	0.358	430	25	Dis	35.441	13118
2	Hen	0.211	430	25	Hen	21.512	13168
2	Hen	0.198	430	25	Hen	22.097	13161
2	Hen	0.217	430	25	Hen	51.677	13226
2	I	0.453	430	25	I	81.007	13100
2	I	0.449	430	25	I	80.896	13147
2	I	0.435	430	25	I	74.162	13113
2	Lozi	0.388	430	25	Lozi	89.444	13079
2	Lozi	0.37	430	25	Lozi	87.481	13075
2	Lozi	0.365	430	25	Lozi	88.124	13007
2	S	0.251	430	25	S	48.567	13153
2	S	0.257	430	25	S	45.368	13221
2	S	0.259	430	25	S	24.819	13278
2	T	0.708	430	25	T	79.53	12960
2	T	0.705	430	25	T	72.315	12966
2	T	0.692	430	25	T	71.877	12901

Table 10: CDABC for FSSLS, Dissipative data set, raw data excerpt.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
1	MT	0.196	701	24	MT	26.136	34085
1	MT	0.203	701	24	MT	23.431	33839
1	MT	0.192	701	24	MT	23.236	33971
1	AC	0.25	701	24	AC	22.519	33949
1	AC	0.252	701	24	AC	23.561	34145
1	AC	0.25	701	24	AC	23.479	34043
1	B	0.564	701	24	B	69.266	33654
1	B	0.57	701	24	B	70.231	33856
1	B	0.559	701	24	B	65.999	33774
1	DL	0.755	701	24	DL	105.283	33682
1	DL	0.752	701	24	DL	85.099	33571
1	DL	1.595	701	24	DL	80.458	33685
1	Dis	0.659	701	24	Dis	24.646	34169
1	Dis	0.553	701	24	Dis	25.706	34223
1	Dis	0.342	701	24	Dis	25.704	34303
1	Hen	0.215	701	24	Hen	25.103	34158
1	Hen	0.237	701	24	Hen	22.915	34297
1	Hen	0.227	701	24	Hen	25.997	34326
1	I	0.445	701	24	I	32.312	34017
1	I	0.412	701	24	I	30.587	34266
1	I	0.413	701	24	I	32.903	33832
1	Lozi	0.373	701	24	Lozi	39.479	34098
1	Lozi	0.383	701	24	Lozi	40.567	34047
1	Lozi	0.377	701	24	Lozi	49.592	33910
1	S	0.256	701	24	S	23.175	34528
1	S	0.267	701	24	S	18.975	34421
1	S	0.275	701	24	S	19.713	34215
1	T	0.705	701	24	T	84.954	33795
1	T	0.705	701	24	T	80.616	33731
1	T	0.703	701	24	T	77.767	33636
2	MT	0.203	621	25	MT	26.873	31587
2	MT	0.215	621	25	MT	29.769	31617
2	MT	0.19	621	25	MT	22.38	31484
2	AC	0.246	621	25	AC	32.748	31658
2	AC	0.265	621	25	AC	30.435	31849
2	AC	0.266	621	25	AC	25.84	31757
2	B	0.56	621	25	B	91.647	31336
2	B	0.586	621	25	B	67.751	31169
2	B	0.562	621	25	B	56.953	31254
2	DL	0.782	621	25	DL	78.94	31306
2	DL	0.774	621	25	DL	79.416	31220
2	DL	0.77	621	25	DL	81.774	31238
2	Dis	0.334	621	25	Dis	27.239	31562
2	Dis	0.334	621	25	Dis	28.064	31626
2	Dis	0.573	621	25	Dis	24.69	31673
2	Hen	0.339	621	25	Hen	22.016	31795
2	Hen	0.339	621	25	Hen	21.887	32205
2	Hen	0.443	621	25	Hen	21.431	31798
2	I	0.817	621	25	I	32.862	31586
2	I	0.778	621	25	I	41.377	31394
2	I	0.423	621	25	I	32.28	31618
2	Lozi	0.385	621	25	Lozi	37.742	31515
2	Lozi	0.386	621	25	Lozi	39.537	31433
2	Lozi	0.388	621	25	Lozi	46.143	31444
2	S	0.274	621	25	S	20.038	31966
2	S	0.265	621	25	S	23.717	31672
2	S	0.282	621	25	S	18.898	31940
2	T	0.706	621	25	T	72.454	31142
2	T	0.709	621	25	T	80.864	31172
2	T	0.716	621	25	T	77.846	31214

4.4 Chaos based DABC for FSSNW

The experimentation conducted on CDABC for Flowshop with no wait constraint is presented in the following text. The entire experimentation was conducted on the Taillard data sets [109], which is a set of 12 data classes of different sizes, each of which contains ten unique instances; therefore a total of 120 data instances.

4.4.1 Experiment setup

The operating parameters of CDABC are given in Table 11. All parameters were kept constant for all the experimentation, in order not to introduce a bias. All experiments were conducted on the machine having Intel i7-3610QM CPU processor running at 2.3GHz with 8GB of RAM. All codes were written in the C programming language.

Table 11: DABC Operating parameters, FSSNW

Parameter	Value
Food Source (FS)	30
Limit (food source)	50
Loop (Local Search)	200
Local search probability (P_L)	0.2
Neighbourhood List (NL)	20
Winning Neighbourhood List (WNL)	0.75 x NL
Iterations	100

For each instance, fourteen (14) repeated experimentations were conducted by each variant of CDABC in order to obtain statistical variance. Therefore, 1680 individual experiments were conducted by each variant, leading to a sum total of 16800 experimentations for all ten variants. The summarized results are presented in Section 5.4. The excerpt from the raw experiment results is given in Table 12. The full experiment results with the schedules can be found on CD (appendix A).

Table 12: CDABC for FSSNW, raw data excerpt.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
1	MT	0.499	15698.000	2	Hen	0.218	17487.000
1	MT	0.452	15674.000	2	Hen	0.219	17560.000
1	MT	0.390	15674.000	2	Hen	0.218	17557.000
1	AC	0.468	15770.000	2	I	0.421	17278.000
1	AC	0.406	15901.000	2	I	0.406	17348.000
1	AC	0.374	15947.000	2	I	0.421	17376.000
1	B	1.154	15674.000	2	Lozi	0.359	17250.000
1	B	1.295	15674.000	2	Lozi	0.374	17426.000
1	B	1.154	15674.000	2	Lozi	0.390	17345.000
1	DL	1.700	15674.000	2	S	0.265	17432.000
1	DL	1.779	15674.000	2	S	0.250	17427.000
1	DL	1.794	15674.000	2	S	0.249	17423.000
1	Dis	0.733	15735.000	2	T	0.686	17253.000
1	Dis	0.718	15747.000	2	T	0.687	17270.000
1	Dis	0.858	15877.000	2	T	0.702	17250.000
1	Hen	0.562	15887.000	120	MT	22.854	15576118.000
1	Hen	0.483	15904.000	120	MT	20.826	15666154.000
1	Hen	0.468	15919.000	120	MT	31.013	15557789.000
1	I	1.030	15745.000	120	AC	23.072	16236131.000
1	I	0.920	15698.000	120	AC	21.310	16077374.000
1	I	0.827	15712.000	120	AC	20.732	15835144.000
1	Lozi	0.562	15712.000	120	B	62.088	15090556.000
1	Lozi	0.546	15788.000	120	B	57.018	14997325.000
1	Lozi	0.421	15698.000	120	B	54.694	15075423.000
1	S	0.266	15832.000	120	DL	75.863	14766976.000
1	S	0.234	15821.000	120	DL	73.819	14692441.000
1	S	0.249	15774.000	120	DL	80.684	14802511.000
1	T	0.655	15674.000	120	Dis	23.883	15668987.000
1	T	0.655	15674.000	120	Dis	28.221	15728190.000
1	T	0.671	15692.000	120	Dis	28.096	15749201.000
2	MT	0.187	17334.000	120	Hen	20.811	16201379.000
2	MT	0.203	17349.000	120	Hen	22.042	15908671.000
2	MT	0.187	17314.000	120	Hen	21.248	15738736.000
2	AC	0.234	17391.000	120	I	31.122	15675062.000
2	AC	0.265	17541.000	120	I	33.447	15525855.000
2	AC	0.250	17317.000	120	I	41.683	15505109.000
2	B	0.561	17300.000	120	Lozi	38.439	15367175.000
2	B	0.546	17323.000	120	Lozi	37.846	15404853.000
2	B	0.531	17270.000	120	Lozi	46.504	15415979.000
2	DL	0.765	17270.000	120	S	18.299	16211489.000
2	DL	0.733	17253.000	120	S	18.548	16114351.000
2	DL	0.749	17270.000	120	S	25.600	16026086.000
2	Dis	0.328	17454.000	120	T	77.064	14766974.000
2	Dis	0.327	17441.000	120	T	73.055	14941783.000
2	Dis	0.312	17284.000	120	T	78.968	14816968.000

4.5 Chaos based DABC for QAP

The experimentation with CDABC solving quadratic assignment problem is presented in this section. The QAP problem dataset used for the experimentation was obtained from the OR Library [9].

4.5.1 Experiment setup

The experimentation were conducted on the Tesla server with the following specifications: Intel Xeon, CPU E5-2640 v2 running at 2GHz with 8 cores and 2GB of cache, hosted at the Media Research Lab [56].

The operating CDABC parameters are given in Table 13. All the parameters were kept fixed for all the simulations in order not to introduce any bias into the experimentations.

Table 13: Operating parameters of CDABC algorithm

Parameter	Value
Food Source (FS)	30
Limit (food source)	50
Loop (Local Search)	200
Local search probability (P_L)	0.2
Neighbourhood List (NL_L)	20
Winning Neighbourhood List (WNL_L)	0.75 x NL_L
Iterations	100

QAP problem dataset on which the experimentation was conducted comprises seventeen different problem instances, with different flow dominance. For each problem instance, fifteen experiments were conducted. Therefore, 255 experiments were conducted for each CDABC variant, leading to a total of 1275 experiments. The example of the raw experiment output data is given in Table 14. The full experiment results with schedules can be found on CD (appendix A). The summary and analysis of results is presented in Section 5.5.

Table 14: CDABC for QAP, raw data excerpt.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
bur26a	B	1.250	5434708.000	bur26e	B	1.260	5388099.000
bur26a	B	1.240	5434266.000	bur26e	B	1.230	5389227.000
bur26a	B	1.250	5433411.000	bur26e	B	1.270	5389258.000
bur26a	DL	1.430	5434761.000	bur26e	DL	1.440	5389361.000
bur26a	DL	1.450	5432106.000	bur26e	DL	1.430	5388735.000
bur26a	DL	1.450	5428840.000	bur26e	DL	1.450	5388682.000
bur26a	Lozi	0.760	5432456.000	bur26e	Lozi	0.780	5389194.000
bur26a	Lozi	0.780	5435114.000	bur26e	Lozi	0.800	5389457.000
bur26a	Lozi	0.750	5438253.000	bur26e	Lozi	0.770	5388229.000
bur26a	MT	0.500	5432953.000	bur26e	MT	0.480	5390077.000
bur26a	MT	0.530	5442118.000	bur26e	MT	0.490	5387722.000
bur26a	MT	0.540	5435803.000	bur26e	MT	0.480	5390558.000
bur26a	T	1.700	5434274.000	bur26e	T	1.730	5388683.000
bur26a	T	1.710	5433864.000	bur26e	T	1.720	5389581.000
bur26a	T	1.670	5433795.000	bur26e	T	1.730	5387929.000
bur26b	B	1.220	3822118.000	bur26f	B	1.400	3782665.000
bur26b	B	1.280	3825837.000	bur26f	B	1.240	3782582.000
bur26b	B	1.250	3818177.000	bur26f	B	1.240	3783125.000
bur26b	DL	1.410	3819110.000	bur26f	DL	1.430	3782740.000
bur26b	DL	1.490	3825036.000	bur26f	DL	1.450	3782694.000
bur26b	DL	1.450	3820506.000	bur26f	DL	1.440	3782779.000
bur26b	Lozi	0.790	3820225.000	bur26f	Lozi	0.790	3782642.000
bur26b	Lozi	0.750	3821108.000	bur26f	Lozi	0.800	3783427.000
bur26b	Lozi	0.800	3826437.000	bur26f	Lozi	0.770	3783120.000
bur26b	MT	0.520	3826062.000	bur26f	MT	0.500	3783248.000
bur26b	MT	0.490	3828199.000	bur26f	MT	0.520	3782993.000
bur26b	MT	0.500	3825380.000	bur26f	MT	0.540	3783454.000
bur26b	T	1.710	3819869.000	bur26f	T	1.700	3782527.000
bur26b	T	1.730	3820521.000	bur26f	T	1.710	3782703.000
bur26b	T	1.690	3825028.000	bur26f	T	1.660	3782480.000
bur26c	B	1.260	5429416.000	bur26g	B	1.250	10120994.000
bur26c	B	1.210	5429545.000	bur26g	B	1.220	10120678.000
bur26c	B	1.220	5430086.000	bur26g	B	1.230	10120623.000
bur26c	DL	1.460	5429579.000	bur26g	DL	1.440	10119051.000
bur26c	DL	1.470	5429456.000	bur26g	DL	1.460	10120600.000
bur26c	DL	1.450	5427731.000	bur26g	DL	1.430	10120772.000
bur26c	Lozi	0.800	5430488.000	bur26g	Lozi	0.780	10121481.000
bur26c	Lozi	0.810	5434209.000	bur26g	Lozi	0.790	10125382.000
bur26c	Lozi	0.810	5431637.000	bur26g	Lozi	0.790	10123368.000
bur26c	MT	0.480	5434833.000	bur26g	MT	0.500	10122060.000
bur26c	MT	0.480	5435117.000	bur26g	MT	0.480	10123729.000
bur26c	MT	0.480	5433763.000	bur26g	MT	0.510	10123012.000
bur26c	T	1.740	5427797.000	bur26g	T	1.710	10121225.000
bur26c	T	1.700	5426955.000	bur26g	T	1.720	10122125.000
bur26c	T	1.770	5429140.000	bur26g	T	1.710	10121358.000
bur26d	B	1.250	3822178.000	bur26h	B	1.290	7098658.000
bur26d	B	1.230	3821827.000	bur26h	B	1.250	7100978.000
bur26d	B	1.250	3822446.000	bur26h	B	1.260	7099509.000
bur26d	DL	1.490	3822461.000	bur26h	DL	1.530	7100807.000
bur26d	DL	1.450	3822061.000	bur26h	DL	1.510	7100115.000
bur26d	DL	1.420	3821885.000	bur26h	DL	1.490	7099216.000
bur26d	Lozi	0.750	3823662.000	bur26h	Lozi	0.760	7100957.000
bur26d	Lozi	0.760	3821687.000	bur26h	Lozi	0.780	7100443.000
bur26d	Lozi	0.820	3822553.000	bur26h	Lozi	0.770	7101166.000
bur26d	MT	0.640	3822211.000	bur26h	MT	0.500	7112147.000
bur26d	MT	0.580	3824521.000	bur26h	MT	0.550	7099421.000
bur26d	MT	0.520	3822869.000	bur26h	MT	0.480	7101725.000
bur26d	T	1.720	3822003.000	bur26h	T	1.680	7098905.000
bur26d	T	1.730	3821645.000	bur26h	T	1.690	7099750.000
bur26d	T	1.700	3821847.000	bur26h	T	1.710	7100418.000

Table 15: CDABC for QAP, raw data excerpt, part 2.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
chr25a	B	1.140	5572.000	tai25b	B	1.140	349144384.000
chr25a	B	1.140	5202.000	tai25b	B	1.130	351333952.000
chr25a	B	1.180	5824.000	tai25b	B	1.130	348650016.000
chr25a	DL	1.320	5452.000	tai25b	DL	1.360	347545376.000
chr25a	DL	1.340	4998.000	tai25b	DL	1.360	349100576.000
chr25a	DL	1.330	5238.000	tai25b	DL	1.350	349108448.000
chr25a	Lozi	0.720	5206.000	tai25b	Lozi	0.740	351248192.000
chr25a	Lozi	0.710	5114.000	tai25b	Lozi	0.740	348984448.000
chr25a	Lozi	0.720	5288.000	tai25b	Lozi	0.730	350668064.000
chr25a	MT	0.490	5896.000	tai25b	MT	0.460	349597824.000
chr25a	MT	0.460	5872.000	tai25b	MT	0.470	357016320.000
chr25a	MT	0.430	5588.000	tai25b	MT	0.450	350177024.000
chr25a	T	1.580	5282.000	tai25b	T	1.550	346665792.000
chr25a	T	1.530	5494.000	tai25b	T	1.580	347447328.000
chr25a	T	1.580	5462.000	tai25b	T	1.560	345230656.000
kra30a	B	1.650	94900.000	tai30b	B	1.690	646048064.000
kra30a	B	1.650	92420.000	tai30b	B	1.640	644814976.000
kra30a	B	1.630	93590.000	tai30b	B	1.640	642084544.000
kra30a	DL	1.880	92770.000	tai30b	DL	1.890	644693120.000
kra30a	DL	1.920	93930.000	tai30b	DL	1.890	652857472.000
kra30a	DL	1.900	93720.000	tai30b	DL	1.910	642384896.000
kra30a	Lozi	1.010	94320.000	tai30b	Lozi	1.030	651466944.000
kra30a	Lozi	1.010	93670.000	tai30b	Lozi	1.070	643570240.000
kra30a	Lozi	1.020	94650.000	tai30b	Lozi	1.020	644374656.000
kra30a	MT	0.640	94480.000	tai30b	MT	0.630	645105152.000
kra30a	MT	0.670	96080.000	tai30b	MT	0.650	645856832.000
kra30a	MT	0.690	95180.000	tai30b	MT	0.690	644553408.000
kra30a	T	2.260	93530.000	tai30b	T	2.250	644099968.000
kra30a	T	2.230	94250.000	tai30b	T	2.250	642299264.000
kra30a	T	2.230	93400.000	tai30b	T	2.230	639882304.000
kra30b	B	1.600	94590.000	tai35b	B	2.280	287764576.000
kra30b	B	1.580	95670.000	tai35b	B	2.320	288945440.000
kra30b	B	1.610	93920.000	tai35b	B	2.330	286918496.000
kra30b	DL	1.930	95410.000	tai35b	DL	2.800	287914720.000
kra30b	DL	1.910	93940.000	tai35b	DL	2.740	286455392.000
kra30b	DL	1.900	93790.000	tai35b	DL	2.740	287806496.000
kra30b	Lozi	1.010	94740.000	tai35b	Lozi	1.480	291141024.000
kra30b	Lozi	1.060	93610.000	tai35b	Lozi	1.470	289470624.000
kra30b	Lozi	1.030	96460.000	tai35b	Lozi	1.490	294444480.000
kra30b	MT	0.670	96490.000	tai35b	MT	1.000	291282944.000
kra30b	MT	0.650	96540.000	tai35b	MT	0.990	293113600.000
kra30b	MT	0.710	97270.000	tai35b	MT	0.990	292140704.000
kra30b	T	2.280	94640.000	tai35b	T	3.130	288337664.000
kra30b	T	2.240	93240.000	tai35b	T	3.160	288305600.000
kra30b	T	2.270	94980.000	tai35b	T	3.190	286219584.000
tai20b	B	0.780	122887368.000	tai40b	B	2.900	683002240.000
tai20b	B	0.740	123130616.000	tai40b	B	3.000	659806912.000
tai20b	B	0.780	123479352.000	tai40b	B	2.950	677257536.000
tai20b	DL	0.880	122516280.000	tai40b	DL	3.530	663156928.000
tai20b	DL	0.900	123362968.000	tai40b	DL	3.580	667944576.000
tai20b	DL	0.900	123778912.000	tai40b	DL	3.570	645325184.000
tai20b	Lozi	0.460	124190568.000	tai40b	Lozi	1.880	664733120.000
tai20b	Lozi	0.460	123314376.000	tai40b	Lozi	1.880	673773056.000
tai20b	Lozi	0.470	123340648.000	tai40b	Lozi	1.990	657746560.000
tai20b	MT	0.310	123573184.000	tai40b	MT	1.280	663268672.000
tai20b	MT	0.320	123737144.000	tai40b	MT	1.240	650451968.000
tai20b	MT	0.300	123975176.000	tai40b	MT	1.260	686733824.000
tai20b	T	1.040	122695464.000	tai40b	T	4.120	664290112.000
tai20b	T	1.040	122684136.000	tai40b	T	4.160	647138304.000
tai20b	T	1.040	122941640.000	tai40b	T	4.150	665227776.000

Table 16: CDABC for QAP, raw data excerpt, part 3.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
tai50b	B	4.540	471272768.000	tai50b	Lozi	2.950	477692928.000
tai50b	B	4.590	472595072.000	tai50b	MT	1.830	484745568.000
tai50b	B	4.520	472204672.000	tai50b	MT	1.950	477929120.000
tai50b	DL	5.360	476478656.000	tai50b	MT	1.940	483437856.000
tai50b	DL	5.390	476339168.000	tai50b	T	6.210	476373440.000
tai50b	DL	5.370	471390400.000	tai50b	T	6.180	476028288.000
tai50b	Lozi	2.950	482963584.000	tai50b	T	6.360	473094144.000
tai50b	Lozi	2.870	481309376.000				

4.6 Chaos based DABC for CVRP

The experimentation on CDABC for the CVRP problem was conducted under the same setup as the one for QAP. CVRP problem dataset, consisting of twelve different instances, was obtained from [110], with the range from 75 to 385 customers. Fifteen experiments were conducted on each instance, each variant having 195 simulations, leading to a total of 975 experiments.

The hardware configuration and CDABC operating parameters are described in the CDABC for QAP experimentation, Section 4.5. The analysis of results is presented in Section 5.6. The example from the experiment output is given in Table 17. The table contains first 3 results out of total 15 experiments conducted for each problem instance an each pseudo-random generator. Only costs and execution times of the problem solutions are shown. The full experiment results with schedules can be found on CD (appendix A).

Table 17: CDABC for CVRP, raw data excerpt.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
tai100a	B	2.130	3036.258	tai100b	DL	2.670	3122.648
tai100a	B	2.140	3063.774	tai100b	DL	2.610	3037.660
tai100a	B	2.170	3129.280	tai100b	DL	2.620	2985.854
tai100a	DL	2.570	3157.669	tai100b	Lozi	1.420	3242.182
tai100a	DL	2.630	3163.348	tai100b	Lozi	1.400	3290.250
tai100a	DL	2.620	3207.618	tai100b	Lozi	1.370	3259.391
tai100a	Lozi	1.520	3452.867	tai100b	MT	0.920	3332.140
tai100a	Lozi	1.430	3493.153	tai100b	MT	0.900	3417.404
tai100a	Lozi	1.430	3407.510	tai100b	MT	0.870	3459.588
tai100a	MT	0.830	3635.807	tai100b	T	3.000	2937.629
tai100a	MT	0.950	3510.925	tai100b	T	3.080	2714.647
tai100a	MT	0.980	3595.376	tai100b	T	3.050	2978.178
tai100a	T	3.020	3077.968	tai100c	B	2.170	2204.639
tai100a	T	3.080	2922.230	tai100c	B	2.150	2180.118
tai100a	T	3.030	3052.822	tai100c	B	2.180	2224.191
tai100b	B	2.190	2930.631	tai100c	DL	2.610	2231.104
tai100b	B	2.190	3026.870	tai100c	DL	2.620	2125.710
tai100b	B	2.090	3147.590	tai100c	DL	2.630	2150.054

Table 18: CDABC for CVRP, raw data excerpt, part 2.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
tai100c	Lozi	1.380	2413.912	tai150c	Lozi	2.100	4973.259
tai100c	Lozi	1.360	2378.772	tai150c	Lozi	2.090	5196.848
tai100c	Lozi	1.400	2394.359	tai150c	Lozi	2.010	4963.071
tai100c	MT	0.920	2461.691	tai150c	MT	1.380	5336.271
tai100c	MT	0.900	2497.000	tai150c	MT	1.310	5487.854
tai100c	MT	0.920	2546.055	tai150c	MT	1.320	5591.835
tai100c	T	3.140	1986.135	tai150c	T	4.540	4121.433
tai100c	T	3.030	2101.921	tai150c	T	4.510	4165.498
tai100c	T	3.050	1925.270	tai150c	T	4.450	4103.235
tai100d	B	2.120	2482.153	tai150d	B	3.160	4944.467
tai100d	B	2.160	2531.926	tai150d	B	3.090	5007.555
tai100d	B	2.150	2488.516	tai150d	B	3.140	5008.102
tai100d	DL	2.700	2441.306	tai150d	DL	3.840	5092.319
tai100d	DL	2.670	2454.727	tai150d	DL	3.730	5161.389
tai100d	DL	2.670	2446.918	tai150d	DL	3.860	4834.228
tai100d	Lozi	1.420	2641.425	tai150d	Lozi	2.040	5460.960
tai100d	Lozi	1.400	2724.867	tai150d	Lozi	2.060	5225.324
tai100d	Lozi	1.460	2661.619	tai150d	Lozi	2.080	5510.790
tai100d	MT	0.930	2820.969	tai150d	MT	1.350	5807.081
tai100d	MT	0.970	2640.192	tai150d	MT	1.350	5718.907
tai100d	MT	0.950	2711.797	tai150d	MT	1.350	5856.792
tai100d	T	3.160	2435.865	tai150d	T	4.410	4801.818
tai100d	T	3.080	2380.239	tai150d	T	4.430	4763.122
tai100d	T	3.150	2383.848	tai150d	T	4.530	4819.787
tai150a	B	3.100	5458.163	tai385	B	8.840	50970.082
tai150a	B	3.140	5022.197	tai385	B	8.730	53868.105
tai150a	B	3.110	5447.435	tai385	B	8.830	53655.703
tai150a	DL	3.830	5437.270	tai385	DL	10.480	54007.609
tai150a	DL	3.910	5697.197	tai385	DL	10.440	54082.902
tai150a	DL	3.820	5543.035	tai385	DL	10.400	53675.328
tai150a	Lozi	2.050	6217.642	tai385	Lozi	5.870	59026.758
tai150a	Lozi	1.990	6220.327	tai385	Lozi	5.730	61179.621
tai150a	Lozi	2.010	5966.238	tai385	Lozi	5.540	61782.418
tai150a	MT	1.320	6444.245	tai385	MT	3.690	64552.980
tai150a	MT	1.390	6348.214	tai385	MT	3.570	66458.000
tai150a	MT	1.380	6221.090	tai385	MT	3.550	64911.215
tai150a	T	4.500	5540.755	tai385	T	12.240	51441.602
tai150a	T	4.500	5464.627	tai385	T	12.220	51118.766
tai150a	T	4.430	5389.713	tai385	T	12.020	51220.199
tai150b	B	3.130	5180.753	tai75a	B	1.680	2146.866
tai150b	B	3.130	5439.010	tai75a	B	1.720	2215.826
tai150b	B	3.100	5006.230	tai75a	B	1.700	2141.800
tai150b	DL	3.810	5063.437	tai75a	DL	2.000	2067.634
tai150b	DL	3.750	4827.841	tai75a	DL	2.050	2167.237
tai150b	DL	3.820	5080.392	tai75a	DL	2.040	2122.669
tai150b	Lozi	2.090	5641.168	tai75a	Lozi	1.080	2171.114
tai150b	Lozi	2.080	5821.550	tai75a	Lozi	1.080	2255.785
tai150b	Lozi	2.080	5662.979	tai75a	Lozi	1.090	2315.647
tai150b	MT	1.340	6202.055	tai75a	MT	0.710	2380.950
tai150b	MT	1.330	5932.041	tai75a	MT	0.740	2363.137
tai150b	MT	1.270	5942.922	tai75a	MT	0.700	2355.405
tai150b	T	4.480	4946.197	tai75a	T	2.380	2108.802
tai150b	T	4.410	5142.101	tai75a	T	2.380	2118.359
tai150b	T	4.420	5054.248	tai75a	T	2.420	2046.926
tai150c	B	3.180	4469.299	tai75b	B	1.660	1755.918
tai150c	B	3.200	4494.129	tai75b	B	1.740	1760.224
tai150c	B	3.120	4788.574	tai75b	B	1.690	1744.971
tai150c	DL	3.870	4285.029	tai75b	DL	2.040	1720.482
tai150c	DL	3.800	4108.254	tai75b	DL	2.040	1736.641
tai150c	DL	3.790	4493.352	tai75b	DL	2.040	1743.031

Table 19: CDABC for CVRP, raw data excerpt, part 3.

Instance	PRNG	Time[s]	Cost	Instance	PRNG	Time[s]	Cost
tai75b	Lozi	1.120	1806.191	tai75c	MT	0.690	1931.137
tai75b	Lozi	1.120	1819.417	tai75c	T	2.430	1648.094
tai75b	Lozi	1.100	1810.348	tai75c	T	2.420	1705.667
tai75b	MT	0.740	1870.131	tai75c	T	2.420	1717.407
tai75b	MT	0.720	1909.485	tai75d	B	1.660	1885.371
tai75b	MT	0.730	1857.421	tai75d	B	1.680	1877.672
tai75b	T	2.430	1662.084	tai75d	B	1.670	1842.354
tai75b	T	2.370	1668.737	tai75d	DL	2.020	1853.000
tai75b	T	2.410	1644.342	tai75d	DL	2.000	1813.701
tai75c	B	1.650	1786.257	tai75d	DL	2.010	1790.526
tai75c	B	1.670	1739.125	tai75d	Lozi	1.030	2029.151
tai75c	B	1.650	1774.615	tai75d	Lozi	1.100	2015.536
tai75c	DL	2.090	1711.402	tai75d	Lozi	1.080	2017.102
tai75c	DL	2.020	1770.101	tai75d	MT	0.700	2071.202
tai75c	DL	2.020	1767.942	tai75d	MT	0.720	2085.977
tai75c	Lozi	1.010	1843.797	tai75d	MT	0.730	2114.971
tai75c	Lozi	1.080	1878.654	tai75d	T	2.440	1785.311
tai75c	Lozi	1.020	1822.863	tai75d	T	2.410	1740.977
tai75c	MT	0.720	1920.342	tai75d	T	2.400	1825.728
tai75c	MT	0.720	1977.484				

4.7 Centralities Based ABC

The following section describes the experimentation conducted for two implementations of the centralities Based ABC, Adaptive ABC 1 and Adaptive ABC 2. The standard test functions for continuous domain optimisation were used as a benchmark. Two sets of experiments were performed: First set of tests was run on small to medium scale problems with the parameter setting described in Section 4.7.2. The second set of tests improves upon the first one, fine tuning the parameters, running on medium to large scale problems as well. It is described in Section 4.7.4.

4.7.1 Experiment set 1: small and medium scale problems

The performances of Adaptive ABC 1 and Adaptive ABC 2 were compared against the performance of the original ABC on small to medium scale problems, optimising the functions of the standard test set: Schwefel, De Jong 1, De Jong 3, De Jong 4, Rosenbrock's Saddle, Rastrigin, Griewangk, Sine Envelope Sine Wave, Ackley One, Ackley Two, Egg Holder, Michalewicz, Master's Cosine Wave, and Schekel's Foxhole, testing for 10, 20 and 30 dimensions.

4.7.2 Experiment set 1 setup

The experiments were conducted again on the server at Media Research Lab (MRL) at the VSB-Technical University of Ostrava, with the following CPU specifications: Intel Xeon CPU E5-2640 v2 with 2.00GHz frequency.

For each of the algorithms, the experiments with population of 15, 30, 45 and 60 solutions were made. The rest of the parameters were set as follows: the termination criterion was 100 generations. The network was evaluated after every 5th generation, with 40% lowest ranking solutions being recreated.

For every algorithm, problem, dimension and every tested value of the number of solutions (15, 30, 45 and 60), 30 experiments were done to obtain the statistically valid sample. This makes up the total amount of $3 \times 168 \times 30 = 15120$ experiments. The statistical analysis of results is described in Section 5.7.1

4.7.3 Experiment set 2: small and medium scale problems

In this experiment set, the Adaptive ABC 1 and Adaptive ABC 2 are again compared to ABC, this time also on medium to large scale problems, optimising the functions of the same test set: Schwefel, De Jong 1, De Jong 3, De Jong 4, Rosenbrock's Saddle, Rastrigin, Griewangk, Sine Envelope Sine Wave, Stretch V Sine Wave, Ackley One, Ackley Two, Egg Holder, Michalewicz, Master's Cosine Wave, and Schekel's Foxhole, testing for 30, 40, 50, 75 and 100 dimensions, in the allowed range of values between -100 to 100 for each dimension, with 15 repetitions for each problem and dimension.

Moreover, the experiments on Adaptive ABC 3, as well as Adaptive ABC 3.b and 3.c were performed, to compare the effect of different centrality measure choice, as well as the effect of elitism incorporated in the pruning of the nodes logic. Five different versions of the complex network analysis incorporating modifications to the ABC algorithm are compared against the canonical variant.

4.7.4 Experiment set 2 setup

The platform used for testing is the same as in the first test set, described in 4.7.2, however, the parameter settings are changed. Extensive experimentation was performed to find the best parameter values of each algorithm. The resulting optimal parameter settings used for each algorithm are given in Table 20.

The excerpt from the raw experiment data comparing the results of ABC, Adaptive ABC 1, Adaptive ABC 2 and Adaptive ABC 3, 3.b and 3.c is shown in the Tables 22, 23, 24, 25, 26 and 27. Due to the space limitations, only selected problems (Schwefel, De Jong 1, Masters Cosine Wave, Shekel's Foxhhole) are included in the tables; for each of these problems and each dimension, the first 3 outputs of 15 repeated experiments are shown. The solutions as well as full experiment results can be found on CD (appendix A). The statistically analysed data is presented in Section 5.7.2.

The next part of the experiment set compares the centrality measures employed in the ABC: Degree centrality, Closeness and Betweenness. The parameter settings used to compare effects of different vertex centrality choice on Adaptive ABC 3, Adaptive ABC 3.b, as well as Adaptive ABC 3.c, are given in Table 21.

The results for different centrality options are compared for each of the three algorithms separately. The conclusions are discussed in the analysis Section 5.7.2. The full output data for this experiment can be found on CD (appendix A).

Table 20: Parameters setting of ABC, Adaptive ABC 1, Adaptive ABC 2, Adaptive ABC 3, 3.b, and 3.c

ABC		**Adaptive ABC 1**		**Adaptive ABC 2**	
Parameter	Value	Parameter	Value	Parameter	Value
Number of Solutions	30	Number of Solutions	30	Number of Solutions	30
Number of Generations	1000	Number of Generations	1000	Number of Generations	1000
Limit	50	Limit	-	Limit	-
CN Gen.Number	-	CN Gen.Number	30	CN Gen.Number	50
CN Cutoff	-	CN Cutoff	0,1	CN Cutoff	0,3
CN Measure	-	CN Measure	-	CN Measure	-
CN Elitism	-	CN Elitism	-	CN Elitism	-
Adaptive ABC 3		**Adaptive ABC 3.b**		**Adaptive ABC 3.c**	
Parameter	Value	Parameter	Value	Parameter	Value
Number of Solutions	30	Number of Solutions	30	Number of Solutions	30
Number of Generations	1000	Number of Generations	1000	Number of Generations	1000
Limit	-	Limit	-	Limit	-
CN Gen.Number	30	CN Gen.Number	15	CN Gen.Number	15
CN Cutoff	0,1	CN Cutoff	0,1	CN Cutoff	0,3
CN Measure	2^2	CN Measure	1^1	CN Measure	2^2
CN Elitism	-	CN Elitism	0,6	CN Elitism	0,6

*1: Closeness
*2: Betweenness

Table 21: Parameters setting of Adaptive ABC 3, 3.b, 3.c

Adaptive ABC 3		**Adaptive ABC 3.b**		**Adaptive ABC 3.c**	
Parameter	Value	Parameter	Value	Parameter	Value
Number of Solutions	30	Number of Solutions	30	Number of Solutions	30
Number of Generations	750	Number of Generations	750	Number of Generations	750
Limit	-	Limit	-	Limit	-
CN Gen.Number[1]	30	CN Gen.Number[1]	15	CN Gen.Number[1]	15
CN Cutoff[2]	0.1	CN Cutoff[2]	0.1	CN Cutoff[2]	0.3
		CN Elitism[3]	0,6	CN Elitism[3]	0,6

*1: Number of generations before the complex network is evaluated
*2: Ratio of solutions to remove on centrality ranking evaluation
*3: Ratio of best solutions to always keep when pruning nodes

Table 22: ABC, raw data excerpt

Problem	Dimension	Cost	Time[s]	Problem	Dimension	Cost	Time[s]
1	10	-636.350	0.14	15	10	-8.336	0.21
1	10	-636.350	0.15	15	10	-8.337	0.21
1	10	-636.350	0.14	15	10	-8.337	0.21
1	20	-1272.700	0.17	15	20	-17.479	0.32
1	20	-1263.467	0.18	15	20	-16.738	0.34
1	20	-1272.700	0.17	15	20	-13.808	0.32
1	30	-1899.817	0.21	15	30	-21.317	0.48
1	30	-1898.642	0.22	15	30	-18.878	0.47
1	30	-1899.817	0.21	15	30	-15.595	0.48
1	40	-2517.699	0.26	15	40	-26.797	0.58
1	40	-2526.929	0.24	15	40	-19.435	0.58
1	40	-2517.682	0.24	15	40	-21.681	0.58
1	50	-3133.489	0.30	15	50	-21.701	0.69
1	50	-3153.679	0.30	15	50	-25.087	0.70
1	50	-3117.643	0.30	15	50	-23.898	0.70
1	100	-6099.227	0.49	15	100	-17.513	1.31
1	100	-6089.252	0.51	15	100	-11.510	1.31
1	100	-6129.309	0.49	15	100	-8.597	1.32
2	10	0.000	0.11	16	10	-1.333	0.55
2	10	0.000	0.11	16	10	-0.448	0.54
2	10	0.000	0.11	16	10	-1.371	0.54
2	20	0.000	0.15	16	20	-0.247	0.93
2	20	0.000	0.19	16	20	-0.247	0.95
2	20	0.000	0.20	16	20	-0.247	0.92
2	30	0.000	0.13	16	30	-0.538	1.34
2	30	0.000	0.11	16	30	-0.538	1.34
2	30	0.000	0.13	16	30	-0.538	1.35
2	40	0.000	0.16	16	40	-0.213	1.76
2	40	0.000	0.16	16	40	-0.214	1.76
2	40	0.000	0.16	16	40	-0.214	1.76
2	75	0.001	0.15	16	75	-0.081	3.14
2	75	0.002	0.14	16	75	-0.075	3.17
2	75	0.000	0.16	16	75	-0.069	3.14
2	100	0.031	0.16	16	100	-0.025	4.16
2	100	0.436	0.17	16	100	-0.025	4.16
2	100	0.006	0.15	16	100	-0.024	4.19

Table 23: Adaptive ABC 1, raw data excerpt

Problem	Dimension	Cost	Time[s]	Problem	Dimension	Cost	Time[s]
1	10	-636.350	0.06	15	10	-8.341	0.12
1	10	-636.350	0.06	15	10	-8.341	0.13
1	10	-636.350	0.07	15	10	-8.342	0.12
1	20	-1272.700	0.10	15	20	-17.563	0.25
1	20	-1272.700	0.11	15	20	-17.537	0.25
1	20	-1272.700	0.10	15	20	-16.745	0.25
1	30	-1899.809	0.14	15	30	-24.100	0.36
1	30	-1889.194	0.14	15	30	-24.706	0.37
1	30	-1890.452	0.15	15	30	-23.120	0.36
1	40	-2498.385	0.18	15	40	-18.266	0.48
1	40	-2508.294	0.18	15	40	-23.558	0.50
1	40	-2515.889	0.18	15	40	-33.905	0.49
1	50	-3065.721	0.22	15	50	-21.106	0.62
1	50	-3100.432	0.23	15	50	-17.432	0.62
1	50	-3112.273	0.23	15	50	-21.512	0.61
1	100	-5740.985	0.44	15	100	-19.811	1.22
1	100	-5823.000	0.45	15	100	-10.746	1.22
1	100	-5665.651	0.45	15	100	-17.157	1.23
2	10	0.000	0.03	16	10	-0.425	0.46
2	10	0.000	0.03	16	10	-1.200	0.48
2	10	0.000	0.04	16	10	-0.577	0.48
2	20	0.000	0.04	16	20	-0.369	0.86
2	20	0.000	0.04	16	20	-0.247	0.86
2	20	0.000	0.04	16	20	-0.248	0.87
2	30	0.000	0.06	16	30	-0.535	1.26
2	30	0.000	0.04	16	30	-0.536	1.26
2	30	0.000	0.05	16	30	-0.526	1.27
2	40	0.000	0.05	16	40	-0.212	1.67
2	40	0.000	0.06	16	40	-0.198	1.68
2	40	0.000	0.06	16	40	-0.204	1.66
2	75	0.682	0.07	16	75	-0.083	3.09
2	75	0.780	0.08	16	75	-0.067	3.08
2	75	0.067	0.08	16	75	-0.073	3.07
2	100	3.300	0.09	16	100	-0.020	4.09
2	100	2.564	0.09	16	100	-0.021	4.08
2	100	0.970	0.09	16	100	-0.023	4.09

Table 24: Adaptive ABC 2, raw data excerpt

Problem	Dimension	Cost	Time[s]	Problem	Dimension	Cost	Time[s]
1	10	-636.350	0.05	15	10	-8.342	0.11
1	10	-636.350	0.04	15	10	-8.342	0.11
1	10	-636.350	0.05	15	10	-8.342	0.11
1	20	-1272.697	0.08	15	20	-14.510	0.23
1	20	-1272.700	0.08	15	20	-16.203	0.22
1	20	-1272.700	0.08	15	20	-13.039	0.22
1	30	-1880.831	0.12	15	30	-16.944	0.34
1	30	-1890.584	0.12	15	30	-20.882	0.34
1	30	-1868.900	0.12	15	30	-19.358	0.34
1	40	-2499.198	0.16	15	40	-16.250	0.46
1	40	-2489.000	0.16	15	40	-13.354	0.47
1	40	-2526.917	0.15	15	40	-14.147	0.45
1	50	-3060.772	0.19	15	50	-16.821	0.58
1	50	-3081.652	0.19	15	50	-15.285	0.57
1	50	-3122.825	0.19	15	50	-17.497	0.58
1	100	-5705.364	0.38	15	100	-10.962	1.17
1	100	-5978.479	0.38	15	100	-9.738	1.17
1	100	-5997.569	0.39	15	100	-10.880	1.16
2	10	0.000	0.01	16	10	-1.251	0.44
2	10	0.000	0.02	16	10	-0.591	0.44
2	10	0.000	0.01	16	10	-0.586	0.45
2	20	0.000	0.02	16	20	-0.411	0.85
2	20	0.000	0.02	16	20	-0.709	0.84
2	20	0.000	0.02	16	20	-0.286	0.85
2	30	0.000	0.02	16	30	-0.525	1.25
2	30	0.000	0.03	16	30	-0.529	1.25
2	30	0.000	0.02	16	30	-0.536	1.24
2	40	0.000	0.03	16	40	-0.214	1.64
2	40	0.000	0.04	16	40	-0.214	1.66
2	40	0.000	0.03	16	40	-0.214	1.65
2	75	0.111	0.05	16	75	-0.092	3.09
2	75	0.037	0.05	16	75	-0.077	3.08
2	75	0.043	0.05	16	75	-0.075	3.09
2	100	36.260	0.06	16	100	-0.047	4.06
2	100	16.531	0.07	16	100	-0.039	4.07
2	100	31.340	0.07	16	100	-0.048	4.07

Table 25: Adaptive ABC 3, raw data excerpt

Problem	Dimension	Cost	Time[s]	Problem	Dimension	Cost	Time[s]
1	10	-636.350	0.05	15	10	-8.341	0.13
1	10	-636.350	0.05	15	10	-8.341	0.12
1	10	-636.350	0.06	15	10	-8.338	0.12
1	20	-1272.700	0.09	15	20	-17.373	0.23
1	20	-1272.700	0.09	15	20	-17.489	0.24
1	20	-1272.700	0.09	15	20	-17.535	0.23
1	30	-1899.523	0.13	15	30	-22.087	0.36
1	30	-1898.931	0.13	15	30	-19.845	0.36
1	30	-1899.811	0.13	15	30	-25.051	0.36
1	40	-2515.410	0.17	15	40	-23.885	0.49
1	40	-2488.747	0.16	15	40	-19.289	0.47
1	40	-2507.744	0.18	15	40	-24.562	0.47
1	50	-3122.912	0.21	15	50	-17.451	0.60
1	50	-3090.737	0.21	15	50	-22.724	0.61
1	50	-3134.273	0.21	15	50	-18.390	0.60
1	100	-5803.864	0.44	15	100	-11.461	1.22
1	100	-5625.739	0.43	15	100	-14.225	1.20
1	100	-5774.195	0.43	15	100	-9.046	1.19
2	10	0.000	0.03	16	10	-0.470	0.45
2	10	0.000	0.02	16	10	-0.468	0.45
2	10	0.000	0.02	16	10	-1.410	0.45
2	20	0.000	0.03	16	20	-0.266	0.85
2	20	0.000	0.03	16	20	-0.295	0.86
2	20	0.000	0.03	16	20	-0.247	0.84
2	30	0.000	0.04	16	30	-0.534	1.26
2	30	0.000	0.03	16	30	-0.527	1.27
2	30	0.000	0.04	16	30	-0.534	1.26
2	40	0.000	0.04	16	40	-0.205	1.67
2	40	0.001	0.05	16	40	-0.210	1.68
2	40	0.000	0.04	16	40	-0.213	1.66
2	75	0.079	0.06	16	75	-0.070	3.09
2	75	0.197	0.07	16	75	-0.059	3.09
2	75	0.171	0.06	16	75	-0.066	3.09
2	100	1.398	0.07	16	100	-0.021	4.10
2	100	2.461	0.08	16	100	-0.023	4.09
2	100	2.691	0.08	16	100	-0.019	4.10

Table 26: Adaptive ABC 3.b, raw data excerpt

Problem	Dimension	Cost	Time[s]	Problem	Dimension	Cost	Time[s]
1	10	-636.350	0.06	15	10	-8.342	0.12
1	10	-636.350	0.06	15	10	-8.342	0.12
1	10	-636.350	0.06	15	10	-8.341	0.12
1	20	-1272.700	0.10	15	20	-17.500	0.25
1	20	-1272.700	0.10	15	20	-17.474	0.24
1	20	-1272.700	0.09	15	20	-17.565	0.24
1	30	-1899.817	0.14	15	30	-23.930	0.37
1	30	-1899.803	0.13	15	30	-22.540	0.35
1	30	-1909.040	0.13	15	30	-21.381	0.36
1	40	-2518.081	0.18	15	40	-20.873	0.48
1	40	-2537.567	0.17	15	40	-20.496	0.49
1	40	-2517.688	0.17	15	40	-31.634	0.49
1	50	-3138.874	0.21	15	50	-16.415	0.60
1	50	-3106.865	0.22	15	50	-18.478	0.61
1	50	-3140.961	0.21	15	50	-18.329	0.61
1	100	-6078.180	0.44	15	100	-18.053	1.19
1	100	-6043.177	0.44	15	100	-12.298	1.19
1	100	-5967.345	0.44	15	100	-19.700	1.23
2	10	0.000	0.03	16	10	-0.470	0.45
2	10	0.000	0.03	16	10	-0.356	0.46
2	10	0.000	0.02	16	10	-0.788	0.44
2	20	0.000	0.03	16	20	-0.250	0.87
2	20	0.000	0.04	16	20	-0.250	0.86
2	20	0.000	0.03	16	20	-0.250	0.87
2	30	0.000	0.04	16	30	-0.530	1.27
2	30	0.000	0.04	16	30	-0.535	1.27
2	30	0.000	0.04	16	30	-0.531	1.26
2	40	0.000	0.05	16	40	-0.214	1.68
2	40	0.000	0.05	16	40	-0.214	1.68
2	40	0.000	0.05	16	40	-0.214	1.68
2	75	0.002	0.07	16	75	-0.092	3.10
2	75	0.002	0.07	16	75	-0.077	3.08
2	75	0.002	0.07	16	75	-0.077	3.10
2	100	0.353	0.09	16	100	-0.024	4.09
2	100	0.026	0.08	16	100	-0.025	4.12
2	100	0.299	0.08	16	100	-0.024	4.12

Table 27: Adaptive ABC 3.c, raw data excerpt

Problem	Dimension	Cost	Time[s]	Problem	Dimension	Cost	Time[s]
1	10	-636.350	0.06	15	10	-8.341	0.12
1	10	-636.350	0.06	15	10	-8.342	0.12
1	10	-636.350	0.06	15	10	-8.341	0.13
1	20	-1272.700	0.10	15	20	-17.546	0.24
1	20	-1272.700	0.10	15	20	-17.528	0.24
1	20	-1272.700	0.10	15	20	-17.399	0.24
1	30	-1899.814	0.14	15	30	-23.178	0.36
1	30	-1890.584	0.14	15	30	-21.467	0.35
1	30	-1890.450	0.14	15	30	-22.856	0.36
1	40	-2517.565	0.18	15	40	-17.957	0.48
1	40	-2526.927	0.18	15	40	-19.895	0.50
1	40	-2499.226	0.19	15	40	-24.203	0.47
1	50	-3133.113	0.22	15	50	-21.026	0.66
1	50	-3126.973	0.22	15	50	-20.584	0.65
1	50	-3133.988	0.22	15	50	-16.330	0.64
1	100	-6085.681	0.48	15	100	-8.966	1.22
1	100	-6063.059	0.48	15	100	-21.422	1.22
1	100	-6061.427	0.48	15	100	-9.054	1.22
2	10	0.000	0.03	16	10	-1.007	0.45
2	10	0.000	0.02	16	10	-0.469	0.46
2	10	0.000	0.03	16	10	-23.667	0.46
2	20	0.000	0.04	16	20	-0.434	0.86
2	20	0.000	0.03	16	20	-0.440	0.87
2	20	0.000	0.04	16	20	-0.343	0.86
2	30	0.000	0.04	16	30	-0.537	1.27
2	30	0.000	0.04	16	30	-0.515	1.27
2	30	0.000	0.05	16	30	-0.527	1.28
2	40	0.000	0.05	16	40	-0.214	1.68
2	40	0.000	0.05	16	40	-0.214	1.68
2	40	0.000	0.05	16	40	-0.214	1.68
2	75	0.006	0.07	16	75	-0.091	3.12
2	75	0.006	0.07	16	75	-0.075	3.11
2	75	0.002	0.08	16	75	-0.075	3.13
2	100	0.114	0.08	16	100	-0.024	4.14
2	100	0.104	0.09	16	100	-0.024	4.15
2	100	0.180	0.09	16	100	-0.025	4.15

5 Analysis of Results

5.1 NEH

The summarised results of the execution time in milliseconds of the ten instances on both platforms are given in Table 28. The first column gives the instance size, the second column is the *average* time for all ten instances of that particular instance size, the third column is the GPU computation time and the final column is the percentage relative difference (PRD) between the CPU and GPU based times computed as in Equation (69). These results were published in [73].

$$PRD = \frac{100 \times (CPU - GPU)}{CPU} \tag{69}$$

For the smaller sized instances of 20 x 5, 20 x 10 and 20 x 20, the CPU version is faster. This is due to the overhead of the data transfer using the PCI Express bus between the CPU and GPU. However, from the medium sized problems (50 x 5) onwards, the GPU variant is faster for all instances. When analysing the PRD values, it becomes obvious that the relative difference is over 100 for all instances over 50 x 20, peaking at 518.75 for the 200 x 10 instance.

Analysing the OTS, the average CPU value is **6215.33** and **1406.68** for the GPU. This gives the average PRD for the twelve sets of **166.46**.

For the ETS, the CPU average value is **186949** and **78808** for the GPU. The average PRD is **150.698** for the four data sets of the ETS.

Overall, the average value is **51398.8** for the CPU and **20757** for the GPU with the cumulative average PRD value of **162.52**.

Therefore, for all the instance sizes barring the first three, GPU has faster execution times. However, statistical tests are needed to verify if there is significant improvement when utilising the GPU in respect to the execution times.

5.1.1 t-test analysis

Paired *t*-test comparison was done on the *raw* time values for each of the data instances. Therefore, the execution time for the ten instances in each set was compared pairwise for the CPU and GPU. The confidence level is 95% for these tests. The *t*-test results for each set are given in Table 29. Column one gives the instance size, column two is the *t*-value (the absolute value is shown), column three gives the *p*-value and the final column outlines the hypothesis. As the confidence level is 95%, we check for *p* values of less that 0.05 for a *not equal* hypothesis to hold true.

For the first three instances, the p values are all negligible ($p < 0.05$), therefore the CPU is significantly faster than the GPU. For the 50 x 5 and 50 x 10 instances, the p values are 0.067 and 0.152 ($p > 0.05$), therefore the hypothesis that the CPU and GPU implementations performances are not significantly different holds.

For all the remaining data sets the p values are all negligible ($p < 0.05$), signifying that the GPU variant is significantly faster than the CPU. As this accounts for all the

Table 28: NEH results

Instances	CPU	GPU	PRD
20 x 5	0	1.64	-100
20 x 10	0	2.79	-100
20 x 20	2	4.77	-58.07
50 x 5	11	8.19	**34.26**
50 x 10	20	15.09	**32.52**
50 x 20	39	16.72	**133.29**
100 x 5	81	30.21	**168.11**
100 x 10	180	57.25	**214.42**
100 x 20	665	112.96	**488.73**
200 x 10	1337	216.08	**518.75**
200 x 20	19719	4748.25	**315.29**
500 x 20	52530	11666.16	**350.28**
500 x 50	53394	19693.05	**171.13**
700 x 20	140090	48446.88	**189.16**
700 x 50	152692	71369.74	**113.95**
1000 x 20	401620	175722.3	**128.55**
Mean	51398.8	**20757**	**162.52**

larger problem dimensions, it verifies the hypothesis of this research. The final row of Table 29 gives the *cumulative* values for all the data sets, which again supports the claim that the GPU implementation is significantly faster than the CPU.

Table 29: NEH *t*-test results

Instances	*t*-value	*p*-value	Hypothesis
20 x 5	416.979	0.000	CPU
20 x 10	960.667	0.000	CPU
20 x 20	1196.56	0.000	CPU
50 x 5	2.079	0.067	Equal
50 x 10	1.56	0.152	Equal
50 x 20	660.52	0.000	GPU
100 x 5	12.39	0.000	GPU
100 x 10	28.26	0.000	GPU
100 x 20	18.41	0.000	GPU
200 x 10	28.71	0.000	GPU
200 x 20	235.73	0.000	GPU
500 x 20	486.79	0.000	GPU
500 x 50	90.01	0.000	GPU
700 x 20	150.77	0.000	GPU
700 x 50	337.32	0.000	GPU
1000 x 20	171.23	0.000	GPU
All	7.125	0.000	**GPU**

5.2 2-opt algorithm

The results of 2-opt CUDA implementation compared to the sequential version are presented in Table 30. The first column is the instance size, the second column gives the average CPU execution time in milliseconds for the three instances of given size, the third column is the average GPU version execution time, the last column is the average PRD for the given dimension, defined in the equation 69 in the NEH results analysis section.

As the Table 30 shows, the PRD value is negative, i.e. in favour of the CPU implementation, for the first three dimensions. For the problems with more than 50 jobs, the PRD value is positive, in favour of the GPU, with the results improving with the dimension of the problem instance, especially for the problems with 100 jobs and more, where the improvement is always greater than 72%, 200 jobs sized problems have the improvement of 92%. It can be therefore stated that the GPU accelerated version is on average faster than the CPU version. However, the *t*-test is needed to prove whether this difference is statistically significant.

Because both algorithms return slightly different results, the analysis of the cost of solutions found by both of them is also included. It could be expected that the results for the GPU accelerated implementation would provide slightly improved results, however the Table 31 shows that this assumption was false. The relative difference is small in all cases, and it doesn't show any increasing or decreasing trend with the change of problem instance dimensions, it is closer to random oscillations.

To find out whether the difference of execution times is significant, the paired *t*-test

Table 30: 2-opt execution time

Instances	CPU	GPU	PRD
20 x 5	15.000	23.493	-70.228
20 x 10	10.000	33.434	-234.340
20 x 20	36.667	65.472	-80.745
50 x 5	700.000	513.004	21.009
50 x 10	943.333	776.344	15.559
50 x 20	1730.000	1173.335	30.448
100 x 5	22260.000	5259.240	76.265
100 x 10	30043.333	5650.605	80.891
100 x 20	32336.667	8878.672	72.541
200 x 10	825756.687	60836.906	92.578
200 x 20	972536.646	67912.173	92.962

Table 31: 2-opt solution cost

Instances	CPU	GPU	PRD
20 x 5	726.500	733.400	-0.959
20 x 10	1119.967	1110.700	0.864
20 x 20	1698.883	1710.667	-0.666
50 x 5	1358.613	1356.640	0.157
50 x 10	1774.653	1745.273	1.620
50 x 20	2546.373	2569.380	-0.893
100 x 5	2519.897	2533.560	-0.538
100 x 10	2993.873	3030.857	-1.258
100 x 20	3936.893	3891.687	1.143
200 x 10	5395.055	5409.933	-0.277
200 x 20	6496.085	6524.793	-0.444

was performed at 95% confidence level. Its results are presented in Table 32. The first column gives again the instance size, the second and third columns present the resulting t-value and p-value of the *t*-test, last column is the conclusion drawn from the *t*-test, with three possible values: CPU where the sequential version is better for given instance size, GPU where the parallel version performs better, or Equal, if there is no significant difference between the performances. For the problems smaller than 20 jobs, both algorithms perform at comparatively same speed for 5 machines, however the sequential implementation is significantly better for problems with 10 and 20 machines. Both sequential and parallel version are performing similarly for the problems with 50 jobs, however for problems with schedules greater than 100 jobs, GPU version performs significantly better than the CPU. The assumption taken from the PRD values is therefore confirmed, the parallel version is significantly faster than the sequential one.

Table 32: 2-opt *t*-test results

Instances	***t*-value**	***p*-value**	**Hypothesis**
20 x 5	2.5091	0.1288	Equal
20 x 10	4.3264	0.0495	CPU
20 x 20	15.756	0.0040	CPU
50 x 5	1.8038	0.2130	Equal
50 x 10	1.0015	0.4221	Equal
50 x 20	2.6927	0.1147	Equal
100 x 5	10.298	0.0093	GPU
100 x 10	11.7225	0.0072	GPU
100 x 20	12.053	0.0068	GPU
200 x 10	16.581	0.0036	GPU
200 x 20	15.777	0.0040	GPU
All	2.7695	0.0093	**GPU**

5.3 Chaos based DABC for FSSLS

The experiment results summary for CDABC applied to the FSSLS is presented in this section. The *average* results obtained by the fifteen experiments are given in Table 33 for the Lozi data sets and Table 34 for the Dissipative data sets. The results were published in [71].

From the average results for the Lozi data sets, Tinkerbell has the lowest average values for 18 data instances. It also has the lowest collective average value of 10241.89. The second best performing variant is the Delayed Logistic with 10252.36 for the collective average value. Mersenne twister is the fifth best performing variant with 10359.71.

As in the Lozi case, Tinkerbell and Delayed Logistic are the two best performing variants in the Dissipative data sets. Tinkerbell obtains 12 best results, whereas Delayed Logistic obtains seven. For the collective average results, Tinkerbell has 14058.66 com-

Table 33: Lozi data sets

	MT Δavg	**Arnold Cat** Δavg	**Burgers** Δavg	**Delayed Logistic** Δavg	**Dissipative** Δavg	**Henon** Δavg	**Ikeda** Δavg	**Lozi** Δavg	**Sinai** Δavg	**c** Tinkerbell Δavg
1	541	541	541	541	541	541	541	541	541	541
2	430	430	430	430	430	430	430	430	430	430
3	502	502	502	502	502	502	502	502	502	502
4	551	551	551	551	551	551	551	551	551	551
5	531	531	531	531	531	531	531	531	531	531
6	1985.8	1997.067	1983.733	**1983.2**	1994.667	2000.6	1986.933	1986.333	1999.333	1983.333
7	2210	2210	2210	2210	2210	2210	2210	2210	2210	2210
8	2105.2	2110.133	2104.533	2103.6	2108.467	2113.667	2105.667	2105.067	2111.133	**2103.333**
9	2174.267	2184.133	2168.2	2165.867	2179.2	2184.067	2172.067	2168.867	2185	**2164.067**
10	2043.667	2048.4	2042.6	2042.467	2047.8	2053.667	2044.6	2043.8	2049.667	**2042.4**
11	28976.867	29239.801	28701.334	28579.133	29114	29248	28977	28954.934	29273	**28611.467**
12	27658.133	27986.533	27440.467	27411.199	27951.199	28036.801	27639.801	27649.199	28109.867	**27333.334**
13	27985	28217.4	27846.268	27755.801	28144.867	28298.666	27999.801	27967.133	28302.268	**27722.4**
14	28839.801	29036.732	28418.732	28443.801	28997	29241.934	28751.801	28735.867	29135.732	**28380.732**
15	28803.801	29060.801	28564.867	28469.4	29039.467	29175.6	28799.732	28696.066	29198.934	**28453.467**
16	8409.134	8556.4	8372.533	8341.667	8541.333	8560.333	8462.066	8430.4	8567.134	**8329.4**
17	8301.733	8393.934	8226.667	8198.6	8362.267	8435.6	8292.866	8295.6	8416.6	**8175.933**
18	8446.733	8522.533	8391.533	8363.934	8521.134	8551.2	8471.467	8435.667	8568.2	**8378.333**
19	8176.933	8287.866	8118.933	8097.333	8261.533	8291	8212.2	8212.6	8286.467	**8093.733**
20	8045	8102.2	7954.267	7933.533	8085	8138.733	8035.4	8012.4	8116.8	**7938.067**
21	12225.733	12370.066	12115.934	12107.333	12307.4	12391.8	12218.066	12198.667	12408.267	**12092.733**
22	12306.533	12466	12234.934	12194.934	12415	12428.333	12335.134	12300.733	12486.6	**12194.4**
23	12364.4	12493.2	12244	12201.333	12470.733	12552.333	12356.467	12361.134	12520.066	**12192.533**
24	12319.6	12500.8	12240	12206.6	12436.6	12537.733	12383.934	12337	12518.733	**12186.2**
25	13059.533	13186.533	12971.066	12944.267	13154.467	13204.8	13105.866	13077.066	13239.934	**12906.467**
Average	10359.71	10461.02	10276.22	10252.36	10435.89	10488.39	10364.63	10349.34	10490.35	**10241.89**

Table 34: Dissipative data sets

	MT Δavg	**Arnold Cat** Δavg	**Burgers** Δavg	**Delayed Logistic** Δavg	**Dissipative** Δavg	**Henon** Δavg	**Ikeda** Δavg	**Lozi** Δavg	**Sinai** Δavg	**Tinkerbell** Δavg
1	701	701	701	701	701	701	701	701	701	701
2	621	621	621	621	621	621	621	621	621	621
3	769	769	769	769	769	769	769	769	769	769
4	743	743	743	743	743	743	743	743	743	743
5	691	691	691	691	691	691	691	691	691	691
6	2230	2230	2230	2230	2230	2230	2230	2230	2230	2230
7	2189.66	2212.93	2188.87	2189.73	2209.67	2217.07	2201.13	2199.93	2216.07	**2188.73**
8	2130.733	2133.467	2129.533	**2128.667**	2132.667	2134.467	2130.867	2130	2130	2128.867
9	2204.2	2215.533	2202.067	**2200.067**	2210.6	2224.667	2205	2205.6	2220.133	2200.2
10	2426.6	2439.73	2418.33	2416	2441.533	2447.333	2426.8	242.467	2441.6	**2412.8**
11	14677.333	14820.934	14601.4	14574.4	14799.467	14856.467	14703.934	14700.333	14839.8	**14547.4**
12	15626.2	15810.267	15569.134	**15491.467**	15762.733	15807.4	15679	15668.134	15806.667	15509.066
13	15115.533	15249.134	15055.134	15057.6	15209.866	15284.8	15158.134	15139.667	15268.733	**15033.267**
14	13565.934	13696.934	13472.533	13473.733	13656.134	13736.066	13578.467	13572.934	13710.2	**13441.134**
15	12959.6	13096.6	12914.866	12863.934	13068.866	13137.733	12998.467	13004.8	13123.467	**12879**
16	21409.934	21560.533	21255.268	21219.666	21512.334	21628.133	21420.666	21375.934	21612.334	**21174.867**
17	20869.666	21028.467	20723.934	**20668.199**	21009.467	21042	20895.6	20840.666	21091.867	20678.801
18	20698.801	20892.801	20542.934	20507.334	20775	20933.268	20696.4	20663.801	20886	**20505.133**
19	21009	21256.334	20875.934	20870.666	21195.867	21298.666	21056.934	21034.666	21335.066	**20832.4**
20	21074.467	21210.533	20917.334	**20863.801**	21198.666	21286.934	21062.732	21032.801	21276	20864
21	31887.133	32214.066	31735.533	**31610.133**	32144.066	32263	31937.934	31915.867	32257.268	31697.533
22	32409	32647.467	32193.467	32130.6	32610.268	32717.533	32418.6	32414.066	32743.334	**32079.801**
23	33096	33305.535	32833.867	32793.133	33255.066	33376.801	33100.266	33049.332	33446.934	**32735.467**
24	33999.734	34190.934	33798.602	**33637.668**	34253.066	34314.332	33996.801	33951.266	34318.734	33657.535
25	31503.334	31811	31282	31236.732	31719.334	31894	31550.334	31509.4	31842.4	**31145.666**
Average	14184.315	14301.928	14098.629	14067.542	14276.787	14334.227	14198.923	14096.266	14332.864	**14058.666**

Table 35: Lozi t-test results

	Tinkerbell		**DL**		**Burgers**		**Lozi**	
	t	p	t	p	t	p	t	p
DL	2.15	**0.032**	-	-	-	-	-	-
Burgers	7.21	**0.00**	5.07	**0.00**	-	-	-	-
Lozi	16.85	**0.00**	15.42	**0.00**	12.9	**0.00**	-	-
MT	14.38	**0.00**	13.98	**0.00**	11.87	**0.00**	1.47	0.141

Table 36: Dissipative t-test results

	Tinkerbell		**DL**		**Burgers**		**Lozi**	
	t	p	t	p	t	p	t	p
DL	2.02	**0.044**	-	-	-	-	-	-
Burgers	21.76	**0.00**	6.64	**0.00**	-	-	-	-
Lozi	8.71	**0.00**	20.67	**0.00**	15.14	**0.00**	-	-
MT	22.77	**0.00**	21.69	**0.00**	17.14	**0.00**	1.41	0.157

pared to 14067.54 for the Delayed Logistic. Once again Mersenne Twister is the fifth best performing variant with the average of 14184.32.

5.3.1 t-test analysis

From the experimentations, the top four performing chaotic systems of Tinkerbell, Delayed Logistics, Burgers and Lozi together with the Mersenne Twister are compared pairwise for their performance. From the results it is obvious that the significant divergence of the results occurs from the medium to large data sets, therefore a comprehensive t-test analysis is conducted from the results of data set of instance 11 to instance 25. As mentioned, 15 experiments have been conducted for each instance by each variant of the algorithm.

The t-test takes all the 15 results for each problem instance by the selected variant and conducts a pairwise comparison. The t and p values for the paired t-test are given in Table 35 for the Lozi test instances and Table 36 for the Dissipative test instances.

The t-tests were conducted at a 95% confidence level, so all pairwise compared variants, which have a value of p of less than 0.05 can be interpreted as being significantly different from each other. From the obtained t-test results (Table 37) all the variants are significantly different from each other apart from Mersenne Twister and Lozi Map. Based on these results, it can be inferred that the hierarchy of the five best performing variants based on average performance are Tinkerbell, Delayed Logistic, Burgers, Lozi and Mersenne Twister for the Lozi data sets. For the Dissipative data sets the best five variants are Tinkerbell, Delayed Logistic, Lozi, Burgers and Mersenne Twister.

The basic premise of this research is therefore achieved as it has been shown that a number of different chaotic systems improves DABC, under the same operating parameters.

Table 37: CDABC, Combined *t*-test results

	Tinkerbell		**DL**		**Burgers**		**Lozi**	
	D	*L*	*D*	*L*	*D*	*L*	*D*	*L*
DL	$\neq$	$\neq$	-	-	-	-	-	-
Burgers	$\neq$	$\neq$	$\neq$	$\neq$	-	-	-	-
Lozi	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	-	-
MT	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$=$	$=$

D = Dissipative data sets
L = Lozi data sets

Table 38: Comparison of CDABC with EDE_C for Lozi Non-Idling results

	EDE_C				**$CDABC_T$**			
Instance	Min	Max	Average	Time (sec)	Min	Max	Average	Time (sec)
10 x 5	425	554	**510.92**	0.89	511	511	511	0.88
20 x 10	2017	2230	2139.68	38.78	2099	2102.2	**2100.62**	3.12
50 x 25	27603	30066	28837.28	494.40	27847.4	28274	**28102.08**	22.11
75 x 30	7912	8584	**8297.76**	773.70	11484.5	11631.83	11561.49	32.2
100 x 50	12039	13191	12393.0	3309.70	12226.6	12387.40	**12314.46**	67.81
Average	9999.20	10925.00	**10435.73**	923.49	10833.7	10981.29	10917.93	25.23

5.3.2 Comparison with Enhanced Differential Evolution

An algorithm comparison is done with the chaos driven Enhanced Differential Evolution (EDE_C) algorithm of [20]. EDE algorithm is an extension of the canonical DE algorithm, with *backward/forward* transformation structure and embedded local search. EDE_C has been shown to significantly improve upon EDE. The comparison between EDE_C and CDABC for the Lozi data sets is given in Table 38. In this case, the Tinkerbell variant of CDABC ($CDABC_T$) is chosen as it is the best performing.

Four different parameters are presented; minimum, maximum, average and execution time for each instance *class*. The instance *class* here refers to the grouping of the problems according to size; 10 x 5, 20 x 10, 50 x 25, 75 x 30 and 100 x 50.

The parameter of most interest is the *average*, as it presents the overall performance of the algorithm. $CDABC_T$ has three better *class* averages of size 20 x 10, 50 x 25 and 100 x 50, whereas EDE_C performs better for the 10 x 5 and 75 x 30. Also, EDE_C has the better cumulative average of 10435.73 compared to 10917.93. However, it is quite obvious that the bias of the 75 x 30 (8297.76 against 11561.49) data class greatly influences the cumulative average in EDE_C favour.

The comparison results between EDE_C and $CDABC_T$ for the Dissipative data sets are given in Table 39. Apart from the 10 x 5 data class, $CDABC_T$ obtains better results for all the remaining data classes, in addition to the cumulative average value of 14058.74 against 14152.97.

Therefore, it can be stated that $CDABC_T$ is a better performing algorithm compared to EDE_C for the non-idling problem.

Table 39: Comparison of CDABC with EDE_C for Dissipative Non-Idling results

	EDE_C				$CDABC_T$			
Instance	Min	Max	Average	Time (sec)	Min	Max	Average	Time (sec)
10 x 5	613	762	**700.60**	0.20	705	705	705	0.72
20 x 10	2145	2494	2276.27	40.32	2228.8	2237.2	**2232.49**	2.62
50 x 25	12897	15936	14517.47	514.40	14181.6	14355.2	**14281.97**	17.36
75 x 30	20437	21958	20931.2	772.3	20686.4	20938	**20811.04**	32.87
100 x 50	30904	33984	32339.33	3401.40	32092	32411	**32263.2**	80
Average	13399.20	15026.80	14152.97	945.62	13978.76	14129.28	**14058.74**	26.71

5.4 Chaos based DABC for FSSNW

The *average* results obtained by the 140 experiments of each problem data class are given in Tables 40 and 41. From the results, it can be concluded that the two most promising results are from the Tinkerbell and Delayed Logistic map systems. Tinkerbell has the best average results for the 20 x 5, 20 x 10, 100 x 5, 100 x 10, 100 x 20, 200 x 10, 200 x 20 and 500 x 20 data sets. Delayed Logistic obtains the best results for the remaining data sets of 20 x 20, 50 x 5, 50 x 10 and 50 x 20 data sets. Additionally, it obtains better cumulative average value and standard deviation. The results were published in [74].

5.4.1 t-test analysis

The paired *t*-test is conducted pairwise on all the different variants of CDABC. All the raw results were used for the computations, implying that for each variant, all 1680 results were pairwise compared. The results comprising of the t and p values is given in Table 42. For all the *t*-test comparisons, the p value is compared to a 95% confidence level. In terms of significance, it can be postulated from the results that all variants of CDABC are significantly different. Therefore, it becomes obvious that the order of the best performing variants is given as Delayed Logistic, Tinkerbell, Burgers, Lozi, Mersenne Twister, Ikeda, Dissipative, Arnold Cat, Sinai and Henon Map.

Table 40: Summarised results for Mersenne Twister, Arnold Cat, Burgers, Delayed Logistic and Dissipative Maps

	MT	Arnold Cat	Burgers	Delayed Logistic	Dissipative
20 x 5	16211.94	16333.94	16188.50	16183.76	16315.27
20 x 10	23560.44	23787.04	23525.30	23520.45	23756.47
20 x 20	38593.95	38857.17	38551.00	**38537.12**	38842.35
50 x 5	84813.39	86367.90	83775.64	**83207.16**	85919.88
50 x 10	119748.21	122404.80	118449.56	**117580.99**	121999.04
50 x 20	175302.27	178829.33	173224.56	**171925.68**	178313.36
100 x 5	332715.06	344226.62	326877.84	324506.33	341968.45
100 x 10	458246.87	475807.93	450901.06	447582.99	472777.88
100 x 20	637540.68	661497.84	628598.67	623943.67	657132.23
200 x 10	1821165.26	1912745.61	1786033.84	1770658.18	1895025.71
200 x 20	2463227.25	2591044.94	2420490.61	2398351.19	2571131.01
500 x 20	15572075.59	16013990.10	15067932.90	14815045.71	15870479.40
Average	1811933.41	1872157.77	1761212.46	**1735920.27**	1856138.42
StdDev	4403079.05	4528691.58	4260041.78	**4188289.04**	4487960.06
Time	5.76	5.60	13.99	19.58	6.47

Table 41: Summarised results for Henon, Ikeda, Lozi, Sinai and Tinkerbell Maps

	Henon	Ikeda	Lozi	Sinai	Tinkerbell
20 x 5	16432.11	16241.76	16233.55	16378.60	**16182.42**
20 x 10	23911.16	23641.86	23622.57	23826.85	**23516.82**
20 x 20	39085.52	38667.38	38646.44	38959.71	38546.31
50 x 5	86881.14	85028.39	84597.73	86821.97	83370.07
50 x 10	123106.13	120354.07	119831.39	122926.35	117754.74
50 x 20	180033.85	175847.83	175407.02	179771.59	172261.97
100 x 5	347755.44	334855.24	333891.74	346144.03	**323948.20**
100 x 10	480264.36	462990.53	460947.43	478851.04	**447040.17**
100 x 20	668043.29	645101.83	641935.18	665638.75	**622663.85**
200 x 10	1933300.76	1847125.13	1837010.47	1929469.41	**1764057.10**
200 x 20	2623881.93	2504658.01	2492349.47	2614873.97	**2391550.34**
500 x 20	16042336.93	15575355.41	15441420.64	16147232.41	**14890741.87**
Average	1880419.38	1819155.62	1805491.14	1887574.56	1740969.49
StdDev	4536745.05	4404156.64	4366160.60	4566556.19	4209720.31
Time	5.67	8.35	9.70	5.15	18.28

Table 42: Paired t-test results: t and p values

	MT		Arnold Cat		Burgers		DL		Disspative		Henon		Ikeda		Lozi		Sinai	
	t	p	t	p	t	p	t	p	t	p	t	p	t	p	t	p	t	p
MT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Arnold Cat	19.58	**0.00**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Burgers	15.2	**0.00**	18.1	**0.00**	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DL	15.42	**0.00**	17.61	**0.00**	14.69	**0.00**	-	-	-	-	-	-	-	-	-	-	-	-
Dissipative	20.49	**0.00**	11.23	**0.00**	11.205	**0.00**	17.61	**0.00**	-	-	-	-	-	-	-	-	-	-
Henon	21.16	**0.00**	6.67	**0.00**	18.84	**0.00**	18.25	**0.00**	15.54	**0.00**	-	-	-	-	-	-	-	-
Ikeda	7.43	**0.00**	17.71	**0.00**	17.37	**0.00**	16.92	**0.00**	17.45	**0.00**	19.33	**0.00**	-	-	-	-	-	-
Lozi	5.17	**0.00**	17.52	**0.00**	17.79	**0.00**	17.13	**0.00**	17.49	**0.00**	18.79	**0.00**	11.82	**0.00**	-	-	-	-
Sinai	19.47	**0.00**	9.95	**0.00**	18.08	**0.00**	17.65	**0.00**	14.82	**0.00**	4.64	**0.00**	17.9	**0.00**	17.68	**0.00**	-	-
Tinkerbell	15.97	**0.00**	18.06	**0.00**	15.93	**0.00**	6.84	**0.00**	18.21	**0.00**	18.72	**0.00**	17.57	**0.00**	17.98	**0.00**	18.07	**0.00**

5.5 Chaos based DABC for QAP

The following text presents the statistical analysis of experiment results. Three different statistical measures of importance; cumulative average, standard deviation of the results and average time, were calculated for each problem instance. These measures for the DABC with Mersenne Twister ($\mathrm{DABC_{MT}}$), $\mathrm{CDABC_B}$, $\mathrm{CDABC_{DL}}$, $\mathrm{CDABC_L}$ and $\mathrm{CDABC_T}$ are given in Table 44. Results of QAP experiment were published as part of [75].

According to the results, the $\mathrm{CDABC_T}$ is the best performing variant of the CDABC algorithm. Of the seventeen instances, it has the better average for thirteen instances. These are spread over the different problem instances of `bur`, `kra`, `tai` and `chr` instance types in the dataset. $\mathrm{CDABC_B}$ map has one better average for the `bur26e` instance, while $\mathrm{CDABC_{DL}}$ has three better average instances of `kra30a`, `bur26b` and `tai40b`.

In terms of cumulative average values of all the seventeen instances, $\mathrm{CDABC_T}$ has better cumulative average of 151351762.78 and the smallest cumulative standard deviation of 1.08E+06. In terms of execution time, $\mathrm{DABC_{MT}}$ has the lowest execution time of 0.67 seconds compared to 2.241 seconds for the $\mathrm{CDABC_T}$.

5.5.1 t-test analysis

Paired *t*-test analysis was conducted on the raw data for the different variants in order to analyse if there is any significant difference between them. All 255 instances for the different variants were pairwise compared and the respective t-value and p-value was calculated. The *t*-test results are given in Table 43.

The variants were tested at 95% confidence level, and from the results only the $\mathrm{CDABC_B}$, $\mathrm{CDABC_{DL}}$ and $\mathrm{CDABC_{DL}}$, $\mathrm{CDABC_T}$ variants are not significantly different. All the other variants are significantly different pairwise.

Based on these results, we can confidently state that chaos maps significantly improve on the Mersenne Twister PRNG in DABC algorithm. In terms of hierarchy obtained from the cumulative average values, the order can be given as $\mathrm{CDABC_T}$, $\mathrm{CDABC_{DL}}$, $\mathrm{CDABC_B}$, $\mathrm{CDABC_L}$ and $\mathrm{DABC_{MT}}$.

Table 43: *t*-test results for the QAP problem instances

	$\mathbf{DABC_{MT}}$		$\mathbf{CDABC_B}$		$\mathbf{CDABC_{DL}}$		$\mathbf{CDABC_L}$	
	t-value	p-value	t-value	p-value	t-value	p-value	t-value	p-value
$\mathbf{CDABC_B}$	5.432	$\mathbf{1.298}e^{-7}$						
$\mathbf{CDABC_{DL}}$	5.6864	$\mathbf{3.555}e^{-8}$	1.667	0.0967				
$\mathbf{CDABC_L}$	2.711	**0.0072**	2.903	**0.0040**	4.4266	$\mathbf{1.0}e^{-5}$		
$\mathbf{CDABC_T}$	7.298	$\mathbf{3.727}e^{-12}$	2.4407	**0.0153**	0.994	0.3207	4.6278	$\mathbf{5.8973}e^{-6}$

Table 44: Average results for the QAP problem instances

Instance	$DABC_{MT}$			$CDABC_{B}$			$CDABC_{DL}$			$CDABC_{L}$			$CDABC_{T}$		
	Average	**Std**	**Time**	**Average**	**Std**	**Time**	**Average**	**Std**	**Time**	**Average**	**Std**	**Time**	**Average**	**Std**	**Time**
bur26a	5435351.47	3.01E+03	0.518	5434405.067	1.27E+03	1.242	5433440.13	1.92E+03	1.447	5434966.73	2.02E+03	0.775	**5433251.93**	2.79E+03	1.696
bur26b	3825372.67	2.88E+03	0.503	3821792.4	2.69E+03	1.247	**3819926.40**	1.94E+03	1.438	3822431.60	3.10E+03	0.779	3820955.67	2.75E+03	1.695
bur26c	5432327.67	2.42E+03	0.509	5429092.933	1.12E+03	1.247	5428807.27	7.51E+02	1.461	5430760.73	1.35E+03	0.786	**5428716.93**	1.18E+03	1.729
bur26d	3824011.60	1.44E+03	0.537	3822009.867	3.68E+02	1.242	3821930.33	3.38E+02	1.459	3822180.13	5.58E+02	0.768	**3821924.87**	2.20E+02	1.724
bur26e	5390063.80	1.79E+03	0.497	**5388476.267**	6.26E+02	1.263	5388615.20	3.38E+02	1.439	5389240.40	1.05E+03	0.786	5388511.40	4.41E+02	1.723
bur26f	3783475.40	4.29E+02	0.511	3782924.733	4.14E+02	1.264	3782829.73	2.10E+02	1.437	3783184.20	3.93E+02	0.773	**3782653.47**	3.19E+02	1.702
bur26g	10125473.87	4.05E+03	0.505	10121200	2.12E+03	1.238	10120160.53	1.22E+03	1.457	10122399.40	1.80E+03	0.775	**10120082.33**	1.30E+03	1.716
bur26h	7101955.53	3.07E+03	0.506	7100231.4	8.52E+02	1.247	7100119.73	5.85E+02	1.504	7100971.07	1.01E+03	0.771	**7099876.00**	5.42E+02	1.698
kra30a	94978.00	5.93E+02	0.669	93702.66667	5.77E+02	1.628	**93458.67**	6.59E+02	1.902	94444.67	6.42E+02	1.021	93534.67	5.03E+02	2.260
kra30b	95906.67	7.82E+02	0.679	94616	6.05E+02	1.611	94158.67	6.26E+02	1.906	94699.33	8.57E+02	1.029	**94076.67**	7.62E+02	2.262
chr25a	5576.80	3.76E+02	0.471	5388	3.31E+02	1.160	5314.93	2.01E+02	1.332	5453.60	2.54E+02	0.718	**5341.20**	1.72E+02	1.572
tai20b	123609163.20	4.63E+05	0.313	123131974.4	2.93E+05	0.769	123150328.00	3.84E+05	0.888	123425646.40	4.74E+05	0.470	**122822634.67**	2.96E+05	1.044
tai25b	351931195.73	2.73E+06	0.474	348934201.6	1.60E+06	1.145	347994845.87	8.41E+05	1.338	349421038.93	2.08E+06	0.729	**347097013.33**	1.49E+06	1.569
tai30b	651098227.20	6.43E+06	0.677	644277290.7	3.74E+06	1.653	643316868.27	3.07E+06	1.914	646544123.73	3.29E+06	1.040	**642278387.20**	1.38E+06	2.221
tai35b	292725510.40	1.61E+06	0.974	288727129.6	1.30E+06	2.338	288440763.73	9.26E+05	2.743	290941043.20	2.00E+06	1.481	**287770026.67**	1.00E+06	3.155
tai40b	667226658.13	1.04E+07	1.241	660323575.5	1.32E+07	2.945	**655818901.33**	1.10E+07	3.551	664875080.53	1.14E+07	1.875	656662830.93	1.01E+07	4.124
tai50b	482051266.13	3.00E+06	1.877	474744234.7	3.95E+06	4.462	473119225.60	4.03E+06	5.393	479978762.67	3.92E+06	2.917	**471260149.33**	4.04E+06	6.209
Average	153750383.19	1.44E+06	**0.67**	152072485	1.42E+06	1.629	151584099.67	1.19E+06	1.918	152958025.14	1.36E+06	1.028	**151351762.78**	**1.08E+06**	2.241

5.6 Chaos based DABC for CVRP

The analysis of experiment results for CDABC solving CVRP problem was performed in the same manner as that of QAP, found in Section 5.5. The same three attributes; average, standard deviation and execution time, were obtained for each problem instance. The tabulated results are given in Table 46. These results were published as part of [75]. For the CVPR problem, $\mathrm{CDABC_T}$ is the best performing variant for all the problem instances, with $\mathrm{CDABC_B}$ the second best performing.

In terms of cumulative average values, $\mathrm{CDABC_T}$ has the best cumulative average value of 6826.437 and standard deviation of 112.973. $\mathrm{DABC_{MT}}$ once again has the lowest execution time of 1.1195 seconds.

5.6.1 t-test analysis

Paired *t*-test was again applied to the different algorithms in order to ascertain if there was significant difference between each two of the variants. The *t*-test results are given in Table 45. At 95% confidence level, only the $\mathrm{CDABC_B}$, $\mathrm{CDABC_{DL}}$ maps are not significantly different, whereas all the other variants are significantly different pairwise. Through the analysis of the cumulative values in Table 46, we can state that the order of best performing variants is $\mathrm{CDABC_T}$, $\mathrm{CDABC_B}$, $\mathrm{CDABC_{DL}}$, $\mathrm{CDABC_L}$ and $\mathrm{DABC_{MT}}$.

Table 45: *t*-test results for the CVRP problem instances

	$\mathrm{DABC_{MT}}$		**$\mathrm{CDABC_B}$**		**$\mathrm{CDABC_{DL}}$**		**$\mathrm{CDABC_L}$**	
	t-value	p-value	t-value	p-value	t-value	p-value	t-value	p-value
$\mathrm{CDABC_B}$	6.023	**8.403**e^{-9}						
$\mathrm{CDABC_{DL}}$	6.399	**1.145**e^{-9}	1.4114	0.159				
$\mathrm{CDABC_L}$	6.043	**7.603**e^{-9}	5.759	**3.268**e^{-8}	6.408	**1.09**e^{-9}		
$\mathrm{CDABC_T}$	6.476	**7.515**e^{-10}	5.63614	**6.061**e^{-8}	5.604	**7.112**e^{-8}	6.523	**5.858**e^{-10}

Table 46: Average results for the CVRP problem instances

Instance	$DABC_{MT}$			$CDABC_{B}$			$CDABC_{DL}$			$CDABC_{L}$			$CDABC_{T}$		
	Average	**Std**	**Time**	**Average**	**Std**	**Time**	**Average**	**Std**	**Time**	**Average**	**Std**	**Time**	**Average**	**Std**	**Time**
tai75a	2342.845	49.762	0.704	2141.405	41.472	1.680	2114.515	53.104	2.037	2222.681	51.401	1.092	**2096.185**	50.166	2.401
tai75b	1869.640	62.235	0.715	1745.197	46.052	1.685	1714.855	39.542	2.037	1816.956	33.931	1.089	**1678.957**	28.920	2.385
tai75c	1938.842	44.641	0.711	1761.108	37.366	1.652	1748.206	41.322	2.049	1843.879	51.041	1.063	**1716.844**	31.109	2.417
tai75d	2109.960	48.108	0.700	1841.167	46.703	1.683	1826.861	49.557	1.995	1983.603	69.140	1.063	**1784.023**	32.802	2.411
tai100a	3516.282	105.034	0.916	3162.238	86.045	2.154	3165.510	80.824	2.607	3349.617	92.724	1.451	**3060.056**	61.203	3.057
tai100b	3408.972	98.248	0.917	3033.001	74.036	2.156	3019.551	86.953	2.623	3250.870	60.472	1.391	**2935.121**	73.899	3.063
tai100c	2498.816	70.574	0.908	2186.654	55.190	2.159	2150.959	47.176	2.619	2380.911	68.574	1.393	**2055.812**	76.071	3.081
tai100d	2729.089	87.405	0.939	2498.172	39.744	2.159	2440.982	69.503	2.679	2616.968	63.197	1.415	**2402.249**	34.891	3.139
tai150a	6450.524	148.854	1.373	5496.065	197.539	3.126	5536.911	160.523	3.835	6097.841	157.913	2.042	**5346.892**	139.102	4.488
tai150b	6125.898	174.801	1.316	5150.960	148.461	3.129	5082.495	187.743	3.809	5756.215	135.314	2.051	**4974.448**	100.401	4.450
tai150c	5302.500	213.994	1.343	4487.373	126.304	3.147	4340.375	139.333	3.832	5012.526	124.796	2.055	**4093.157**	131.397	4.491
tai150d	5769.406	142.592	1.330	4970.953	75.503	3.143	4957.906	131.110	3.817	5405.630	125.025	2.089	**4780.173**	114.187	4.455
tai385	64679.206	1026.345	3.664	52628.353	1314.226	8.732	53691.375	1003.324	10.487	60089.740	1243.935	5.660	**51819.760**	594.504	12.181
Average	8364.768	174.815	**1.195**	7007.896	176.049	2.816	7060.808	160.770	3.417	7832.880	175.189	1.835	**6826.437**	**112.973**	4.001

5.7 Centralities Based ABC

This part presents the statistical analysis of the results of experiments conducted on the Centralities Based ABC, described in Section 4.7. The first part contains the analysis performed on the first test set (described in Section 4.7.1), with mean, standard deviation and execution time for each problem and dimension. These results are published in [72]. The second part consists of results of the second test set (presented in Section 4.7.3), with the same statistical summary for each problem and its selected dimension, together with the *t*-test analysis of differences between algorithms. Finally the results of three selected vertex centrality measures are statistically compared.

5.7.1 Analysis of experiment set 1

The experiment results are shown in Tables 47, 48 and 49. Each table contains data for one problem dimension setting (10, 20 and 30 variables). Each table row contains experiment data for ABC, Adaptive ABC 1 and Adaptive ABC 2: mean of the best solution values found for the problem of given dimension, average standard deviation of the best solution costs, and the average time needed by the algorithm, calculated over the results with different number of solutions settings (15, 30, 45 and 60 number of solutions). The best value of each experiment is marked in bold for visual comparison of the algorithm results.

For problems of size 10, ABC has found best values for 5 problems, Adaptive ABC 1 has better results for 6 problems, Adaptive ABC 2 has found only 3 best values. The average of standard deviation for all problems is better for ABC, with the value of 718.210, the second best for Adaptive ABC 1, with the value of 1123.011, and worst for Adaptive ABC 2, 1574.011.

With the problems size 20, ABC has 4 best values, Adaptive ABC 1 has 6 best values again, while Adaptive ABC 2 has 4 best values. The total average of standard deviations is best for Adaptive ABC 1, with the value of 1539920.718, ABC has second best aver-

Table 47: Experiments results, problems with 10 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
10	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-3880.080	116.334	0.025	**-3907.749**	106.029	0.006	-3891.357	106.473	0.006
De Jong One	**0.014**	0.205	0.019	0.028	0.137	0.002	0.018	0.624	0.002
De Jong Three	0.317	0.381	0.019	**0.299**	0.323	0.002	0.321	0.874	0.002
De Jong Four	0.003	521.083	0.027	**0.001**	0.676	0.011	0.000	16.199	0.010
Rosenbrock	1928.039	9.138E+03	0.022	1276.045	1.533E+04	0.005	**1260.884**	2.164E+04	0.005
Rastrigin	-1859.894	88.054	0.024	-1863.108	79.435	0.007	**-1868.017**	95.773	0.007
Griewangk	0.119	0.071	0.025	0.125	0.073	0.008	**0.116**	0.086	0.008
Sine Envelope Sine Wave	-12.172	0.382	0.032	**-12.269**	0.395	0.017	-12.212	0.403	0.018
Ackley One	**-35.943**	0.950	0.031	-35.854	0.961	0.011	-35.776	0.856	0.012
Ackley Two	158.680	8.515	0.035	**158.636**	8.593	0.016	161.207	10.592	0.016
Egg Holder	**-3721.439**	179.570	0.030	-3698.499	196.476	0.011	-3658.960	161.040	0.011
Michalewicz	-13.657	0.589	0.053	**-14.020**	0.544	0.033	-13.906	0.665	0.036
Masters Cosine Wave	**-0.297**	0.280	0.034	-0.273	0.288	0.015	-0.229	0.214	0.015
Shekels Foxhole	**-0.844**	0.243	0.079	-0.809	0.272	0.060	-0.584	0.191	0.060

Table 48: Experiments results, problems with 20 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
20	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**
Schwefel	-6888.910	206.794	0.028	**-6920.429**	186.889	0.011	-6903.745	210.672	0.011
De Jong One	95.523	459.192	0.020	**56.656**	996.295	0.003	110.205	1648.798	0.002
De Jong Three	**58.780**	47.833	0.020	72.478	46.505	0.002	80.503	48.015	0.002
De Jong Four	1.189E+04	3.938E+07	0.036	7.688E+03	2.099E+07	0.020	**4.169E+03**	6.209E+07	0.019
Rosenbrock	**8.467E+04**	5.550E+05	0.025	1.581E+05	5.347E+05	0.009	9.392E+04	4.046E+06	0.010
Rastrigin	**-1.149E+03**	2.830E+04	0.030	4.502E+02	2.850E+04	0.014	1.884E+03	4.568E+04	0.013
Griewangk	1.019	0.272	0.032	**0.979**	0.340	0.016	1.027	0.415	0.016
Sine Envelope Sine Wave	**-18.008**	1.138	0.050	-17.753	1.096	0.033	-18.070	1.088	0.033
Ackley One	-64.300	3.311	0.043	-64.007	3.526	0.026	**-64.740**	2.997	0.024
Ackley Two	365.382	7.028	0.051	**363.916**	7.529	0.033	364.175	7.255	0.033
Egg Holder	-6581.499	276.558	0.042	-6589.035	268.476	0.023	**-6628.405**	285.912	0.023
Michalewicz	-24.753	0.964	0.090	**-24.887**	1.014	0.070	-24.640	1.044	0.073
Masters Cosine Wave	-0.192	0.203	0.049	**-0.272**	0.284	0.030	-0.257	0.198	0.030
Shekels Foxhole	-0.161	0.014	0.133	-0.159	0.015	0.119	**-0.215**	0.095	0.115

Table 49: Experiments results, problems with 30 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
30	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**
Schwefel	**-9355.025**	316.083	0.033	-9136.969	332.937	0.017	-9103.915	340.394	0.017
De Jong One	1.785E+04	2.675E+04	0.020	1.706E+04	2.015E+04	0.004	**1.642E+04**	2.802E+04	0.003
De Jong Three	547.536	207.305	0.020	568.659	179.005	0.004	**546.905**	178.496	0.003
De Jong Four	**6.894E+07**	5.083E+09	0.045	1.114E+08	2.997E+09	0.029	1.433E+08	7.780E+09	0.028
Rosenbrock	**9.752E+05**	3.222E+06	0.029	1.238E+06	1.167E+07	0.013	1.123E+06	6.189E+07	0.013
Rastrigin	8.485E+05	1.337E+06	0.037	1.122E+06	1.699E+06	0.020	**6.427E+05**	1.349E+06	0.020
Griewangk	5.665	7.051	0.040	**4.393**	5.106	0.024	4.736	5.163	0.024
Sine Envelope Sine Wave	-21.167	1.003	0.067	-20.961	1.170	0.050	**-21.277**	1.178	0.050
Ackley One	**-87.692**	5.600	0.056	-86.643	6.379	0.037	-86.256	5.472	0.036
Ackley Two	567.736	6.181	0.068	563.594	6.892	0.051	**561.884**	7.050	0.050
Egg Holder	**-9141.888**	397.819	0.055	-8704.746	352.142	0.036	-8704.871	366.210	0.035
Michalewicz	**-33.760**	1.535	0.127	-33.591	1.565	0.110	-33.326	1.477	0.108
Masters Cosine Wave	-0.246	0.259	0.066	-0.479	0.294	0.047	**-0.527**	0.312	0.053
Shekels Foxhole	-0.294	0.134	0.189	-0.421	0.022	0.170	**-0.423**	0.018	0.181

age standard deviation value of 2854647.978, Adaptive ABC 2 has the largest standard deviation of 4727753.998.

For the problems size 30, ABC has 5 best values, Adaptive ABC 1 has only 1 best value, while Adaptive ABC 2 has 7 best values. Average standard deviation of Adaptive ABC 1 has the lowest value of the three algorithms, with the value of 215051695.164, the second best is ABC again, with the value of 363408745.264, and the worst value holds Adaptive ABC 2, with 560248294.495.

For the overall analysis, on the 10 and 20 sized problem, Adaptive ABC 1 has better performance than both ABC and Adaptive ABC 2. However, for the larger 30 dimension problem, Adaptive ABC 2 has a marked better performance than Adaptive ABC 1. We can conclude that the ensemble population is more efficient for higher dimensional problems.

5.7.2 Analysis of experiment set 2

The results for the second experiment set, are presented in Tables 50 - 58. Each table contains results for one problem dimension (10, 20, 30, 40, 50, 75, 100) reached by ABC and five centralities based ABC modifications: Adaptive ABC 1, 2, 3, 3.b and 3.c. Each row contains data for one of the test functions, the last row contains summary: the average for the dimension. The mean of the costs of best solutions found, the standard deviation of costs, and the average execution time are presented for every algorithm. Table 59 shows the average values across all dimensions for each problem, with the last row showing total mean of each algorithm.

From these results, it can be observed that the Adaptive ABC 3.b has the best overall average. The 2nd best average was achieved by the Adaptive ABC 3.c. The Adaptive ABC 3, Adaptive ABC 1 as well as Adaptive ABC 2 have all worse average than the original ABC. Comparing by the individual testing problems, The Adaptive ABC 3.b is better than the original ABC in ten out of total 15 problems. The Adaptive ABC 3.c is better in nine problems, Adaptive ABC 3 in three problems. The Adaptive ABC 1 improves upon ABC in only two of the problems, whereas Adaptive ABC 2 gives four better results than ABC. The results suggest that the algorithms are capable of finding the solutions of different quality. This hypothesis is confirmed by the pairwise comparison of the algorithms using *t*-test, presented in Table 61. Accordingly, all the algorithms results are statistically significantly different at 95% confidence level, apart from the Adaptive ABC 1 and Adaptive ABC 3. Considering this analysis, it can be stated that the Adaptive ABC 3.b and Adaptive ABC 3.c both significantly improve upon the original ABC algorithm, as well as the other centralities based ABC versions, showing the necessity to incorporate the elitism in the low centrality nodes removal logic.

Table 50: Experiments results, experiment set 2, problems with 10 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
10	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-636.350	0.000	0.145	-636.350	0.000	0.065	-636.350	0.000	0.044
De Jong One	0.000	0.000	0.110	0.000	0.000	0.033	0.000	0.000	0.013
De Jong Three	0.000	0.000	0.108	0.000	0.000	0.035	0.000	0.000	0.015
De Jong Four	0.000	0.000	0.177	0.000	0.000	0.098	0.000	0.000	0.084
Rosenbrock	2.133	1.899	0.130	1.017	0.961	0.058	17.487	20.802	0.037
Rastrigin	-2000.000	0.000	0.139	-2000.000	0.000	0.065	-2000.000	0.000	0.041
Griewangk	0.003	0.005	0.151	0.001	0.003	0.078	0.005	0.007	0.049
Sine Envelope Sine Wave	-13.398	0.028	0.213	-13.430	0.020	0.135	-13.460	0.004	0.109
Stretch V Sine Wave	9.037	0.026	0.302	9.013	0.008	0.215	9.013	0.012	0.203
Ackley One	-39.350	0.156	0.178	-39.650	0.077	0.103	-39.734	0.071	0.081
Ackley Two	0.000	0.000	0.194	0.000	0.000	0.121	0.000	0.000	0.091
Egg Holder	-1117.474	82.880	0.175	-1151.976	96.639	0.097	-1010.963	106.928	0.078
Michalewicz	-14.790	0.448	0.375	-16.236	0.400	0.264	-16.944	0.387	0.233
Masters Cosine Wave	-8.338	0.003	0.209	-8.341	0.001	0.125	-8.342	0.000	0.109
Shekels Foxhole	-0.641	0.386	0.539	-3.048	9.656	0.471	-0.733	0.242	0.444
Total	-254.611	560.401	0.210	-257.267	563.997	0.131	-246.668	550.642	0.109
Problem	**Adaptive ABC 3**			**Adaptive ABC 3.b**			**Adaptive ABC 3.c**		
10	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**
Schwefel	-636.350	0.000	0.053	-636.350	0.000	0.058	-636.350	0.000	0.059
De Jong One	0.000	0.000	0.023	0.000	0.000	0.027	0.000	0.000	0.025
De Jong Three	0.000	0.000	0.022	0.000	0.000	0.029	0.000	0.000	0.027
De Jong Four	0.000	0.000	0.085	0.000	0.000	0.089	0.000	0.000	0.087
Rosenbrock	0.784	0.782	0.045	0.291	0.426	0.050	1.059	1.044	0.051
Rastrigin	-2000.000	0.000	0.053	-2000.000	0.000	0.058	-2000.000	0.000	0.059
Griewangk	0.001	0.003	0.065	0.001	0.003	0.068	0.004	0.006	0.070
Sine Envelope Sine Wave	-13.430	0.018	0.121	-13.460	0.002	0.121	-13.456	0.006	0.123
Stretch V Sine Wave	9.010	0.007	0.204	9.005	0.003	0.209	9.009	0.010	0.209
Ackley One	-39.739	0.059	0.091	-39.813	0.015	0.095	-39.811	0.020	0.097
Ackley Two	0.000	0.000	0.110	0.000	0.000	0.119	0.000	0.000	0.120
Egg Holder	-1126.979	101.240	0.089	-1201.023	92.494	0.088	-1298.046	58.198	0.090
Michalewicz	-16.149	0.385	0.245	-16.884	0.338	0.250	-16.960	0.293	0.254
Masters Cosine Wave	-8.340	0.002	0.123	-8.341	0.000	0.120	-8.341	0.001	0.124
Shekels Foxhole	-0.615	0.307	0.449	-0.620	0.305	0.453	-3.252	6.550	0.455
Total	-255.454	561.482	0.119	-260.480	569.295	0.122	-267.076	580.102	0.123

Table 51: Experiments results, experiment set 2, problems with 20 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
20	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-1272.084	2.384	0.176	-1272.700	0.000	0.103	-1271.682	2.821	0.081
De Jong One	0.000	0.000	0.177	0.000	0.000	0.042	0.000	0.000	0.020
De Jong Three	0.000	0.000	0.114	0.000	0.000	0.043	0.000	0.000	0.021
De Jong Four	0.000	0.000	0.271	0.000	0.000	0.187	0.000	0.000	0.167
Rosenbrock	6.082	7.982	0.161	11.130	8.301	0.089	44.236	42.362	0.066
Rastrigin	-8000.000	0.000	0.175	-7994.610	14.051	0.107	-7977.992	24.626	0.077
Griewangk	0.001	0.003	0.194	0.001	0.003	0.125	0.009	0.014	0.095
Sine Envelope Sine Wave	-27.853	0.147	0.337	-27.872	0.189	0.264	-28.140	0.171	0.229
Stretch V Sine Wave	19.526	0.223	0.522	19.264	0.136	0.444	19.125	0.089	0.420
Ackley One	-79.492	0.528	0.266	-80.250	0.402	0.190	-81.023	0.318	0.164
Ackley Two	0.000	0.000	0.317	0.000	0.000	0.266	1.334	5.165	0.207
Egg Holder	-1879.144	160.573	0.259	-1899.041	186.893	0.181	-1788.733	171.041	0.153
Michalewicz	-26.649	0.799	0.613	-29.323	1.067	0.550	-32.498	1.342	0.485
Masters Cosine Wave	-16.975	0.931	0.329	-17.165	0.536	0.251	-15.814	2.351	0.222
Shekels Foxhole	-0.247	-	0.936	-0.410	0.202	0.862	-0.479	0.186	0.845
Total	-751.789	2016.188	0.323	-752.732	2015.828	0.247	-742.111	2008.543	0.217

Table 52: Experiments results, experiment set 2, problems with 20 variables, part 2

Problem	Adaptive ABC 3			Adaptive ABC 3.b			Adaptive ABC 3.c		
20	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-1272.699	0.003	0.091	-1272.643	0.218	0.097	-1272.699	0.001	0.101
De Jong One	0.000	0.000	0.029	0.000	0.000	0.036	0.000	0.000	0.035
De Jong Three	0.000	0.000	0.029	0.000	0.000	0.035	0.000	0.000	0.035
De Jong Four	0.000	0.000	0.163	0.000	0.000	0.161	0.000	0.000	0.231
Rosenbrock	9.571	11.037	0.077	5.444	7.987	0.081	7.301	6.968	0.082
Rastrigin	-7999.851	0.245	0.096	-8000.000	-	0.099	-7998.501	5.714	0.103
Griewangk	0.002	0.006	0.110	0.001	0.005	0.114	0.001	0.003	0.119
Sine Envelope Sine Wave	-27.854	0.158	0.242	-28.275	0.067	0.243	-28.245	0.072	0.245
Stretch V Sine Wave	19.245	0.106	0.423	19.094	0.045	0.423	19.140	0.061	0.433
Ackley One	-80.467	0.424	0.184	-81.528	0.190	0.182	-81.553	0.141	0.191
Ackley Two	0.000	0.001	0.256	0.000	0.000	0.242	0.000	0.000	0.271
Egg Holder	-1934.294	98.779	0.175	-2084.353	151.101	0.173	-2034.980	182.218	0.185
Michalewicz	-29.364	0.667	0.510	-32.765	0.654	0.519	-32.888	1.026	0.533
Masters Cosine Wave	-17.478	0.138	0.235	-17.347	0.424	0.241	-17.472	0.118	0.239
Shekels Foxhole	-0.424	0.206	0.852	-0.313	0.137	0.861	-0.458	0.190	0.863
Total	-755.574	2018.004	0.231	-766.179	2024.227	0.234	-762.690	2021.952	0.244

Table 53: Experiments results, experiment set 2, problems with 30 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
30	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-1899.829	5.823	0.213	-1891.821	8.886	0.141	-1895.748	12.000	0.117
De Jong One	0.000	0.000	0.119	0.000	0.000	0.049	0.000	0.000	0.026
De Jong Three	0.000	0.000	0.123	0.000	0.000	0.049	0.000	0.000	0.027
De Jong Four	0.000	0.000	0.313	0.000	0.000	0.255	0.000	0.000	0.328
Rosenbrock	7.091	6.130	0.192	56.020	27.892	0.120	891.509	2414.158	0.097
Rastrigin	-17918.654	42.112	0.219	-17838.674	90.979	0.160	-17706.274	148.238	0.119
Griewangk	0.001	0.003	0.239	0.000	0.000	0.173	0.010	0.014	0.142
Sine Envelope Sine Wave	-41.855	0.398	0.462	-41.469	0.395	0.386	-42.395	0.259	0.351
Stretch V Sine Wave	30.532	0.545	0.740	30.168	0.437	0.669	29.550	0.377	0.641
Ackley One	-118.513	0.716	0.360	-119.038	0.930	0.283	-121.172	0.687	0.245
Ackley Two	0.008	0.009	0.438	0.091	0.071	0.382	0.060	0.160	0.325
Egg Holder	-2578.375	149.720	0.351	-2583.173	161.748	0.273	-2492.603	244.105	0.232
Michalewicz	-38.473	1.059	0.869	-40.707	1.210	0.802	-45.407	1.182	0.760
Masters Cosine Wave	-20.540	2.717	0.479	-22.755	2.031	0.365	-18.478	3.682	0.339
Shekels Foxhole	-0.538	0.000	1.340	-0.531	0.004	1.264	-0.529	0.004	1.247
Total	-1505.276	4462.560	0.430	-1496.793	4443.991	0.358	-1426.765	4475.488	0.333

Problem	Adaptive ABC 3			Adaptive ABC 3.b			Adaptive ABC 3.c		
30	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-1894.065	8.524	0.129	-1902.369	4.256	0.133	-1897.410	7.584	0.140
De Jong One	0.000	0.000	0.035	0.000	0.000	0.042	0.000	0.000	0.043
De Jong Three	0.001	0.002	0.035	0.000	0.000	0.042	0.000	0.000	0.042
De Jong Four	0.000	0.000	0.225	0.000	0.000	0.251	0.000	0.000	0.241
Rosenbrock	49.613	33.469	0.106	16.246	15.356	0.112	25.853	18.293	0.115
Rastrigin	-17814.239	84.260	0.142	-17896.761	63.903	0.144	-17846.948	59.390	0.151
Griewangk	0.001	0.004	0.157	0.002	0.004	0.162	0.001	0.003	0.168
Sine Envelope Sine Wave	-41.693	0.437	0.363	-42.806	0.168	0.361	-42.666	0.130	0.370
Stretch V Sine Wave	30.103	0.398	0.644	29.474	0.199	0.643	29.376	0.106	0.654
Ackley One	-119.623	0.691	0.272	-122.196	0.371	0.276	-122.348	0.259	0.285
Ackley Two	0.175	0.175	0.370	0.015	0.011	0.379	0.019	0.013	0.386
Egg Holder	-2592.100	186.541	0.263	-2843.781	193.088	0.259	-2749.223	175.292	0.267
Michalewicz	-40.836	1.031	0.776	-46.402	1.074	0.787	-46.531	1.094	0.808
Masters Cosine Wave	-22.527	2.311	0.359	-22.570	1.575	0.360	-22.862	1.983	0.359
Shekels Foxhole	-0.530	0.004	1.265	-0.532	0.003	1.267	-0.526	0.007	1.271
Total	-1496.381	4438.026	0.343	-1522.112	4461.946	0.348	-1511.551	4448.014	0.353

Table 54: Experiments results, experiment set 2, problems with 40 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
40	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-2517.117	9.747	0.250	-2503.965	10.151	0.183	-2500.462	19.317	0.155
De Jong One	0.000	0.000	0.151	0.000	0.000	0.057	0.000	0.000	0.033
De Jong Three	0.000	0.000	0.127	0.009	0.004	0.056	0.000	0.000	0.033
De Jong Four	0.000	0.000	0.391	0.000	0.000	0.309	0.000	0.000	0.291
Rosenbrock	39.843	26.153	0.221	218.971	173.120	0.151	494.042	1366.079	0.127
Rastrigin	-31525.824	154.295	0.299	-31181.696	423.923	0.207	-30982.978	347.975	0.161
Griewangk	0.000	0.001	0.282	0.002	0.006	0.221	0.019	0.026	0.191
Sine Envelope Sine Wave	-55.809	0.347	0.583	-54.010	0.757	0.515	-56.341	0.389	0.470
Stretch V Sine Wave	42.491	1.076	0.960	41.787	0.697	0.893	40.281	0.500	0.864
Ackley One	-157.886	1.228	0.459	-156.374	1.203	0.383	-160.301	0.690	0.334
Ackley Two	0.677	1.432	0.576	18.608	29.257	0.518	10.940	16.145	0.455
Egg Holder	-3318.308	225.714	0.437	-3145.190	180.445	0.367	-3134.971	381.104	0.317
Michalewicz	-49.489	1.634	1.181	-51.854	2.046	1.085	-57.625	1.516	1.007
Masters Cosine Wave	-21.497	4.397	0.582	-24.676	4.111	0.488	-16.535	2.551	0.459
Shekels Foxhole	-0.214	0.000	1.763	-0.205	0.009	1.669	-0.214	0.000	1.654
Total	-2504.209	7837.340	0.551	-2455.906	7755.884	0.473	-2424.276	7720.585	0.437
Problem	**Adaptive ABC 3**			**Adaptive ABC 3.b**			**Adaptive ABC 3.c**		
40	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**
Schwefel	-2495.867	10.913	0.169	-2521.670	10.573	0.173	-2516.884	9.947	0.179
De Jong One	0.000	0.000	0.043	0.000	0.000	0.048	0.000	0.000	0.050
De Jong Three	0.020	0.017	0.041	0.003	0.001	0.048	0.006	0.002	0.049
De Jong Four	0.000	0.000	0.312	0.000	0.000	0.299	0.000	0.000	0.305
Rosenbrock	232.207	293.744	0.139	64.557	41.240	0.145	93.681	40.516	0.144
Rastrigin	-31161.675	182.743	0.191	-31575.427	202.676	0.192	-31379.175	210.523	0.200
Griewangk	0.003	0.009	0.209	0.001	0.002	0.209	0.000	0.000	0.217
Sine Envelope Sine Wave	-54.387	0.703	0.487	-56.990	0.276	0.482	-56.752	0.271	0.489
Stretch V Sine Wave	41.514	0.600	0.871	39.983	0.261	0.872	39.861	0.208	0.873
Ackley One	-157.120	1.271	0.365	-162.161	0.523	0.365	-162.621	0.448	0.377
Ackley Two	13.165	12.467	0.500	1.011	2.134	0.501	2.075	2.745	0.528
Egg Holder	-3210.244	242.317	0.357	-3560.113	261.769	0.351	-3395.376	226.408	0.361
Michalewicz	-51.179	1.919	1.051	-59.052	1.202	1.076	-58.881	1.342	1.157
Masters Cosine Wave	-22.050	2.673	0.477	-24.214	3.143	0.484	-22.413	2.908	0.486
Shekels Foxhole	-0.209	0.003	1.667	-0.214	0.000	1.680	-0.214	0.000	1.681
Total	-2457.721	7751.292	0.459	-2523.619	7851.813	0.462	-2497.113	7802.460	0.473

Table 55: Experiments results, experiment set 2, problems with 50 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
50	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-3123.266	14.246	0.299	-3095.192	17.098	0.225	-3097.403	21.520	0.190
De Jong One	0.000	0.000	0.133	0.002	0.002	0.063	0.000	0.001	0.039
De Jong Three	0.005	0.002	0.133	0.086	0.056	0.063	0.012	0.007	0.039
De Jong Four	0.000	0.000	0.463	0.000	0.000	0.378	0.000	0.000	0.417
Rosenbrock	100.526	31.196	0.252	297.664	76.785	0.183	2806.078	3958.395	0.155
Rastrigin	-48832.261	421.571	0.318	-47927.307	603.606	0.258	-47483.690	603.225	0.204
Griewangk	0.000	0.001	0.329	0.001	0.002	0.282	0.017	0.028	0.234
Sine Envelope Sine Wave	-69.333	0.707	0.713	-65.727	1.432	0.636	-69.513	0.954	0.589
Stretch V Sine Wave	54.373	1.412	1.179	53.532	1.654	1.120	51.191	0.972	1.084
Ackley One	-194.727	1.755	0.550	-194.109	1.847	0.481	-197.792	1.927	0.420
Ackley Two	21.957	21.054	0.709	111.246	69.892	0.645	104.373	72.654	0.577
Egg Holder	-4012.241	184.559	0.533	-3798.274	219.280	0.541	-3746.387	299.382	0.404
Michalewicz	-59.872	1.466	1.426	-60.770	2.040	1.371	-66.490	2.236	1.300
Masters Cosine Wave	-21.297	4.711	0.697	-22.271	3.577	0.612	-15.985	1.707	0.573
Shekels Foxhole	-0.152	0.002	2.157	-0.145	0.005	2.074	-0.154	0.001	2.075
Total	-3742.419	12139.305	0.659	-3646.751	11920.769	0.595	-3447.716	11923.355	0.553

Table 56: Experiments results, experiment set 2, problems with 50 variables, part 2

Problem	Adaptive ABC 3			Adaptive ABC 3.b			Adaptive ABC 3.c		
50	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-3101.361	21.389	0.211	-3133.884	13.132	0.214	-3127.692	10.627	0.221
De Jong One	0.004	0.005	0.049	0.000	0.000	0.055	0.000	0.000	0.057
De Jong Three	0.103	0.049	0.047	0.020	0.008	0.055	0.028	0.013	0.056
De Jong Four	0.000	0.000	0.364	0.000	0.000	0.385	0.000	0.000	0.389
Rosenbrock	284.579	61.605	0.171	191.262	92.129	0.173	261.840	260.836	0.176
Rastrigin	-47380.703	1100.680	0.244	-48949.783	305.220	0.240	-48832.091	283.851	0.249
Griewangk	0.004	0.006	0.259	0.003	0.007	0.261	0.004	0.006	0.270
Sine Envelope Sine Wave	-66.334	0.801	0.601	-70.865	0.348	0.605	-70.358	0.319	0.612
Stretch V Sine Wave	53.720	1.309	1.089	50.951	0.400	1.089	50.850	0.469	1.091
Ackley One	-194.352	2.140	0.471	-201.279	1.144	0.459	-201.654	0.515	0.471
Ackley Two	134.501	67.664	0.701	36.158	24.927	0.680	33.897	22.297	0.657
Egg Holder	-3804.238	152.934	0.451	-4233.409	199.861	0.446	-4127.928	233.697	0.453
Michalewicz	-61.221	2.668	1.373	-70.632	1.457	1.335	-71.102	1.224	1.369
Masters Cosine Wave	-22.202	5.374	0.603	-22.121	3.672	0.603	-20.920	3.536	0.649
Shekels Foxhole	-0.143	0.005	2.065	-0.154	0.000	2.082	-0.154	0.001	2.105
Total	-3610.509	11787.159	0.580	-3760.249	12170.542	0.579	-3740.352	12142.721	0.588

Table 57: Experiments results, experiment set 2, problems with 75 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
75	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-4620.687	21.731	0.397	-4508.942	62.615	0.335	-4556.735	45.743	0.285
De Jong One	0.004	0.008	0.147	0.315	0.266	0.077	0.101	0.164	0.052
De Jong Three	0.100	0.031	0.148	0.917	0.320	0.079	1.810	0.662	0.053
De Jong Four	0.000	0.000	0.631	1.179	1.288	0.550	0.009	0.022	0.525
Rosenbrock	742.134	1088.059	0.326	1922.262	2012.711	0.257	3023.516	3252.994	0.231
Rastrigin	-105471.576	2360.415	0.458	-99711.179	4407.692	0.387	-99264.805	2439.901	0.319
Griewangk	0.012	0.018	0.460	0.032	0.022	0.405	0.036	0.030	0.355
Sine Envelope Sine Wave	-102.096	0.872	1.026	-92.573	3.088	0.947	-99.291	1.407	0.893
Stretch V Sine Wave	85.238	2.410	1.734	87.972	2.278	1.681	80.588	1.831	1.665
Ackley One	-290.211	1.681	0.791	-281.130	1.704	0.729	-290.058	2.223	0.641
Ackley Two	361.269	72.932	1.025	825.968	162.436	1.003	543.344	71.133	0.913
Egg Holder	-5556.169	280.671	0.758	-5094.241	162.544	0.692	-5305.720	394.125	0.621
Michalewicz	-86.173	2.223	2.112	-83.218	1.555	2.053	-92.659	3.526	1.947
Masters Cosine Wave	-14.811	5.104	1.012	-16.225	2.815	0.916	-12.114	2.687	0.888
Shekels Foxhole	-0.073	0.003	3.155	-0.070	0.005	3.096	-0.078	0.004	3.086
Total	-7663.536	26266.198	0.945	-7129.929	24891.518	0.881	-7064.804	24792.273	0.832

Problem	Adaptive ABC 3			Adaptive ABC 3.b			Adaptive ABC 3.c		
75	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-4493.870	46.963	0.321	-4623.439	21.427	0.319	-4631.283	17.215	0.327
De Jong One	0.139	0.116	0.062	0.003	0.002	0.069	0.005	0.003	0.072
De Jong Three	1.379	0.429	0.061	0.444	0.276	0.069	0.666	0.421	0.071
De Jong Four	0.969	1.748	0.537	0.000	0.000	0.544	0.000	0.000	0.549
Rosenbrock	2712.752	3141.650	0.239	1143.278	2616.116	0.247	2286.333	3204.940	0.249
Rastrigin	-99718.991	3063.781	0.370	-108019.527	955.871	0.364	-106741.358	1164.227	0.375
Griewangk	0.039	0.030	0.391	0.027	0.018	0.392	0.074	0.045	0.401
Sine Envelope Sine Wave	-94.543	1.874	0.907	-103.493	0.737	0.907	-102.766	0.520	0.927
Stretch V Sine Wave	85.853	3.053	1.635	79.857	1.049	1.613	79.219	1.315	1.637
Ackley One	-282.736	4.719	0.726	-296.128	1.011	0.703	-296.779	0.819	0.709
Ackley Two	735.518	118.994	0.954	412.803	91.171	1.023	393.818	88.343	1.003
Egg Holder	-5274.513	324.605	0.685	-5795.373	299.128	0.672	-5821.754	270.313	0.688
Michalewicz	-83.692	2.655	2.008	-97.675	1.952	2.003	-98.104	1.592	2.028
Masters Cosine Wave	-14.509	3.665	0.894	-18.474	3.120	0.957	-17.008	5.094	0.909
Shekels Foxhole	-0.066	0.005	3.088	-0.077	0.004	3.093	-0.077	0.004	3.120
Total	-7095.085	24905.463	0.858	-7821.185	26910.742	0.865	-7663.268	26623.112	0.871

Table 58: Experiments results, experiment set 2, problems with 100 variables

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
100	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-6065.883	39.698	0.497	-5808.159	148.183	0.449	-5939.358	107.243	0.383
De Jong One	0.114	0.183	0.163	1.552	1.026	0.091	19.829	14.768	0.066
De Jong Three	2.003	0.953	0.164	7.761	3.418	0.091	25.018	8.291	0.069
De Jong Four	0.238	0.588	0.811	323.571	777.647	0.724	20.940	43.492	0.699
Rosenbrock	3253.219	3488.047	0.400	5473.917	9315.859	0.333	42471.642	36667.925	0.307
Rastrigin	-181320.642	4378.645	0.566	-157050.500	7863.902	0.515	-152976.188	6953.837	0.444
Griewangk	0.112	0.074	0.592	0.200	0.086	0.568	0.568	0.165	0.487
Sine Envelope Sine Wave	-133.137	1.605	1.357	-113.243	5.503	1.261	-126.302	2.128	1.188
Stretch V Sine Wave	121.535	3.696	2.289	127.227	4.127	2.244	114.610	3.283	2.211
Ackley One	-381.091	3.416	1.036	-366.874	5.340	0.980	-380.029	4.423	0.883
Ackley Two	912.167	91.084	1.347	1493.455	132.331	1.309	1154.430	64.877	1.212
Egg Holder	-7336.796	327.040	0.991	-6617.322	315.740	0.927	-6958.360	454.621	0.837
Michalewicz	-110.275	2.348	2.773	-102.574	2.869	2.835	-115.436	4.243	2.689
Masters Cosine Wave	-12.145	3.625	1.315	-12.990	3.091	1.225	-10.130	1.779	1.161
Shekels Foxhole	-0.024	0.000	4.173	-0.021	0.001	4.085	-0.049	0.005	4.123
Total	-12738.040	45248.494	1.232	-10842.933	39373.495	1.176	-8179.921	41393.700	1.117

Problem	Adaptive ABC 3			Adaptive ABC 3.b			Adaptive ABC 3.c		
100	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-5814.047	123.322	0.431	-6070.419	44.113	0.439	-6079.374	22.168	0.481
De Jong One	3.691	3.920	0.077	0.116	0.092	0.083	0.319	0.596	0.087
De Jong Three	9.600	2.188	0.076	6.813	3.112	0.085	6.742	3.363	0.084
De Jong Four	87.816	105.746	0.705	0.064	0.100	0.713	0.330	0.503	0.740
Rosenbrock	8337.907	5181.084	0.313	4100.912	4831.517	0.321	1833.999	2289.607	0.325
Rastrigin	-160669.949	9153.976	0.493	-188552.947	2120.697	0.492	-184303.983	3245.157	0.503
Griewangk	0.205	0.101	0.534	0.198	0.145	0.527	0.309	0.128	0.541
Sine Envelope Sine Wave	-118.613	5.267	1.201	-132.984	1.217	1.210	-131.547	0.858	1.223
Stretch V Sine Wave	123.901	5.530	2.182	109.350	1.624	2.184	110.000	1.992	2.200
Ackley One	-368.154	4.474	0.969	-388.725	2.084	0.955	-389.758	1.287	0.964
Ackley Two	1453.792	108.445	1.287	914.010	104.068	1.367	879.744	88.211	1.336
Egg Holder	-6673.646	332.578	0.917	-7233.107	251.396	0.908	-7347.405	240.981	0.927
Michalewicz	-105.765	3.983	2.694	-122.046	2.801	2.755	-123.740	2.471	2.722
Masters Cosine Wave	-11.144	3.460	1.204	-13.491	3.240	1.208	-13.034	4.569	1.223
Shekels Foxhole	-0.022	0.001	4.099	-0.024	0.000	4.107	-0.024	0.000	4.163
Total	-10916.295	40323.118	1.146	-13158.819	47071.352	1.157	-13037.162	45948.709	1.168

Table 59: Experiments results, experiment set 2, summary

Problem	ABC			Adaptive ABC 1			Adaptive ABC 2		
	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-2876.459	1779.407	0.282	-2816.733	1698.659	0.214	-2842.534	1738.518	0.179
De Jong One	0.017	0.078	0.143	0.267	0.664	0.059	2.847	8.825	0.036
De Jong Three	0.301	0.781	0.131	1.253	2.968	0.059	3.834	9.231	0.037
De Jong Four	0.034	0.231	0.437	46.393	307.139	0.357	2.993	17.574	0.359
Rosenbrock	593.004	1746.413	0.240	1140.140	3974.329	0.170	7106.930	19933.507	0.146
Rastrigin	-56438.422	60639.845	0.311	-51957.709	52953.777	0.243	-51198.847	51729.135	0.195
Griewangk	0.018	0.048	0.321	0.034	0.076	0.265	0.095	0.204	0.222
Sine Envelope Sine Wave	-63.354	39.258	0.670	-58.332	32.954	0.592	-62.206	37.031	0.547
Stretch V Sine Wave	51.819	36.857	1.104	52.709	38.872	1.038	49.194	34.531	1.013
Ackley One	-180.181	112.001	0.520	-176.775	106.901	0.450	-181.444	111.226	0.396
Ackley Two	185.154	325.856	0.658	349.910	552.262	0.606	259.212	412.973	0.540
Egg Holder	-3685.501	2026.048	0.500	-3469.888	1766.817	0.440	-3491.105	1944.057	0.377
Michalewicz	-55.103	31.296	1.335	-54.954	28.138	1.280	-61.008	31.869	1.203
Masters Cosine Wave	-16.515	5.834	0.661	-17.775	6.070	0.569	-13.914	4.169	0.536
Shekels Foxhole	-0.270	0.259	2.009	-0.633	3.683	1.932	-0.319	0.268	1.925
Total	-4165.697	4449.614	0.621	-3797.473	4098.221	0.552	-3361.752	5067.541	0.514

Table 60: Experiments results, experiment set 2, summary, part 2

Problem	Adaptive ABC 3			Adaptive ABC 3.b			Adaptive ABC 3.c		
	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-2815.466	1697.860	0.201	-2880.111	1780.789	0.205	-2880.242	1784.551	0.215
De Jong One	0.548	1.933	0.045	0.017	0.053	0.051	0.046	0.246	0.053
De Jong Three	1.586	3.420	0.045	1.040	2.635	0.052	1.063	2.650	0.052
De Jong Four	12.684	49.555	0.342	0.009	0.043	0.349	0.047	0.218	0.363
Rosenbrock	1661.059	3643.777	0.156	788.856	2461.075	0.161	644.295	1711.451	0.163
Rastrigin	-52392.201	54021.330	0.227	-57856.349	63065.321	0.227	-57014.579	61676.072	0.234
Griewangk	0.037	0.080	0.246	0.033	0.087	0.248	0.056	0.118	0.255
Sine Envelope Sine Wave	-59.551	34.530	0.560	-64.125	39.278	0.561	-63.684	38.817	0.570
Stretch V Sine Wave	51.907	37.727	1.007	48.245	33.009	1.005	48.208	33.105	1.014
Ackley One	-177.456	107.358	0.440	-184.547	114.170	0.434	-184.932	114.506	0.442
Ackley Two	333.879	526.378	0.597	194.857	331.038	0.616	187.079	317.795	0.614
Egg Holder	-3516.573	1806.711	0.420	-3850.166	1970.302	0.414	-3824.959	1998.860	0.424
Michalewicz	-55.458	29.053	1.237	-63.637	34.163	1.246	-64.029	34.616	1.267
Masters Cosine Wave	-16.893	6.136	0.556	-18.080	5.808	0.568	-17.436	5.829	0.570
Shekels Foxhole	-0.287	0.257	1.926	-0.277	0.244	1.935	-0.672	2.633	1.951
Total	-3798.146	4131.074	0.534	-4258.949	4655.868	0.538	-4211.316	4514.764	0.546

Table 61: Centralities Based ABC, experiment set 2, *t*-test results

	ABC		Adaptive ABC 3		Adaptive ABC 3.b		Adaptive ABC 3.c		Adaptive ABC 1	
	p	*different*	*p*	*different*	*p*	*different*	*p*	*different*	*p*	*different*
ABC	-		-		-		-		-	
Adaptive ABC 3	0.00000	T	-		-		-		-	
Adaptive ABC 3.b	0.00062	T	0.00000	T	-		-		-	
Adaptive ABC 3.c	0.01396	T	0.00000	T	0.02133	T	-		-	
Adaptive ABC 1	0.00000	T	0.49274	F	0.00000	T	0.00000	T	-	
Adaptive ABC 2	0.00000	T	0.00027	T	0.00000	T	0.00000	T	0.00055	T

Comparing the standard deviations of results, the least varied outcomes are provided by Adaptive ABC 1. The 2nd is Adaptive ABC 3, followed by ABC, Adaptive ABC 3.c, 3.b with Adaptive ABC 2 coming last. This is nearly the reversed order with respect to the overall best mean values. However, comparing by problem, none of the algorithms improves upon the original ABC standard deviation in more than 50% of the problems, nor is it worse in more than 60% of the problems.

The execution times comparison shows that the fastest running algorithm was the Adaptive ABC 2, followed by Adaptive ABC 3, 3.b, 3.c and 1, with the ABC being the slowest. This is however probably heavily influenced by the settings, such as the number of generations between complex network analysis and subsequent nodes removal and recreation, the cut-off ratio parameter, defining how many nodes are removed, as well as the original ABC parameter limit, defining how many attempts are made to improve the solution, before it is removed from the population and randomly reinitialized.

The statistical analysis of the results for different centrality measures (degree, closeness and betweenness) is presented in the following text. The Tables 62 - 64 show the overall averaged results across all the dimensions for each problem in the row, the different vertex centralities are arranged in the columns. Each Table contains data for one of the algorithms of Adaptive ABC 3, Adaptive ABC 3.b and Adaptive ABC 3.c separately. Again the average best solution cost, standard deviation and the average execution time are presented.

For the Adaptive ABC 3, the overall best average value is achieved by the closeness centrality. For the Adaptive ABC 3.b, the best average value is obtained by degree centrality, in the case of Adaptive ABC 3.c, the closeness centrality achieves the best average result again. The second best was the degree, closeness and betweenness for the discussed algorithms, respectively. Comparing the differences by the problem, the closeness centrality holds the best average results in seven out of 15 problems, the degree centrality in five and the betweenness in three problems. For the Adaptive ABC 3.b, the degree centrality has six best values, while the closeness reaches only three bests, with betweenness achieving six bests. In the case of Adaptive ABC 3.c, the closeness finds eight best values, the degree four and the betweenness only three best values. Neither these results, nor the comparison by the problem dimension, suggest significant difference, which is again confirmed by the t-tests, presented in Table 65 (separately for each algorithm). None of the centrality measures shows statistically significant difference compared to the other.

Table 62: Experiments results, Centrality Comparison, Adaptive ABC 3

Problem	0 (Degree)			1 (Closeness)			2 (Betweenness)		
	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-2790.246	1664.502	0.144	-2809.482	1693.009	0.154	-2804.769	1687.863	0.152
De Jong One	0.590	2.007	0.027	0.893	2.896	0.035	1.312	3.481	0.035
De Jong Three	4.304	9.901	0.026	4.446	10.036	0.034	4.233	9.469	0.034
De Jong Four	133.600	905.509	0.251	135.665	894.607	0.258	329.602	2755.210	0.256
Rosenbrock	2372.851	5100.782	0.108	2380.672	7002.190	0.116	2801.224	7185.797	0.117
Rastrigin	-51659.233	53055.774	0.162	-52031.825	53673.736	0.170	-51274.297	52365.238	0.171
Griewangk	0.128	0.261	0.178	0.147	0.295	0.187	0.142	0.285	0.188
Sine Envelope Sine Wave	-58.655	33.739	0.421	-59.577	34.581	0.420	-58.593	33.460	0.420
Stretch V Sine Wave	53.502	39.930	0.744	53.010	39.077	0.752	53.373	39.675	0.754
Ackley One	-175.577	105.882	0.321	-176.614	107.074	0.328	-176.380	106.518	0.328
Ackley Two	386.034	560.437	0.446	384.246	547.581	0.447	376.987	540.080	0.441
Egg Holder	-3404.949	1723.898	0.306	-3412.709	1742.880	0.313	-3427.275	1749.004	0.315
Michalewicz	-53.967	27.600	0.924	-54.974	28.770	0.912	-54.595	28.397	0.939
Masters Cosine Wave	-13.348	4.560	0.419	-13.200	4.624	0.419	-12.629	4.954	0.457
Shekels Foxhole	-0.281	0.242	1.438	-0.355	0.724	1.442	-0.312	0.311	1.439
Total	-3680.350	4215.668	0.394	-3706.644	4385.472	0.399	-3616.132	4433.983	0.403

Table 63: Experiments results, Centrality Comparison, Adaptive ABC 3.b

Problem	0 (Degree)			1 (Closeness)			2 (Betweenness)		
	Mean	STD	Time	Mean	STD	Time	Mean	STD	Time
Schwefel	-2849.868	1745.241	0.144	-2848.316	1748.593	0.154	-2850.962	1747.104	0.156
De Jong One	0.698	2.806	0.028	0.774	2.684	0.040	0.526	1.756	0.041
De Jong Three	4.766	11.289	0.027	4.133	9.666	0.039	4.591	12.547	0.040
De Jong Four	1.670	8.007	0.249	4.622	24.404	0.269	3.105	11.437	0.261
Rosenbrock	1176.996	3190.986	0.110	1311.549	3815.466	0.121	2623.685	14424.012	0.123
Rastrigin	-54967.182	57940.332	0.163	-54945.678	57881.489	0.175	-55338.811	58732.955	0.175
Griewangk	0.151	0.298	0.180	0.151	0.302	0.192	0.156	0.308	0.197
Sine Envelope Sine Wave	-62.248	37.059	0.419	-62.226	36.967	0.424	-62.185	36.995	0.426
Stretch V Sine Wave	49.557	34.768	0.747	49.654	34.874	0.757	49.509	34.557	0.749
Ackley One	-181.870	111.737	0.318	-181.781	111.527	0.328	-181.714	111.322	0.329
Ackley Two	299.957	447.506	0.438	290.355	437.018	0.448	290.733	429.886	0.450
Egg Holder	-3694.459	1902.999	0.303	-3704.719	1861.701	0.311	-3730.288	1941.187	0.312
Michalewicz	-60.330	32.176	0.911	-60.171	32.016	0.936	-60.237	32.142	0.924
Masters Cosine Wave	-13.469	4.529	0.420	-13.105	5.053	0.424	-13.144	4.588	0.420
Shekels Foxhole	-0.271	0.231	1.436	-0.331	0.773	1.447	-0.394	1.044	1.445
Total	-4019.727	4364.664	0.393	-4010.339	4400.169	0.404	-3951.029	5168.123	0.403

Table 64: Experiments results, Centrality Comparison, Adaptive ABC 3.c

Problem	0 (Degree)			1 (Closeness)			2 (Betweenness)		
	Mean	**STD**	**Time**	**Mean**	**STD**	**Time**	**Mean**	**STD**	**Time**
Schwefel	-2851.353	1745.876	0.145	-2852.574	1751.344	0.156	-2849.086	1748.295	0.157
De Jong One	1.192	4.126	0.028	1.237	4.226	0.039	0.990	3.044	0.040
De Jong Three	5.526	12.420	0.028	6.046	13.721	0.039	5.660	13.298	0.040
De Jong Four	5.212	22.210	0.261	28.654	179.495	0.269	15.367	104.223	0.264
Rosenbrock	3742.319	24154.197	0.113	1694.053	4819.157	0.123	2220.658	7746.102	0.124
Rastrigin	-53971.929	56350.917	0.167	-54100.916	56657.816	0.179	-54133.571	56721.115	0.181
Griewangk	0.155	0.292	0.185	0.164	0.295	0.197	0.156	0.295	0.197
Sine Envelope Sine Wave	-61.821	36.670	0.416	-61.835	36.664	0.428	-61.923	36.823	0.429
Stretch V Sine Wave	49.523	34.814	0.753	49.310	34.431	0.762	49.473	34.677	0.771
Ackley One	-182.309	112.028	0.324	-182.569	112.322	0.332	-182.474	112.088	0.333
Ackley Two	296.006	440.482	0.453	303.366	443.322	0.472	308.344	452.719	0.466
Egg Holder	-3724.222	1896.295	0.309	-3755.579	1950.235	0.322	-3710.892	1918.440	0.321
Michalewicz	-61.013	32.632	0.928	-61.038	32.931	0.951	-60.694	32.447	0.937
Masters Cosine Wave	-12.990	4.584	0.427	-13.274	4.658	0.432	-12.810	4.315	0.429
Shekels Foxhole	-0.328	0.329	1.449	-0.342	0.328	1.457	-0.298	0.262	1.456
Total	-3784.402	5656.525	0.399	-3929.686	4402.730	0.411	-3894.073	4595.210	0.410

Table 65: Centralities Based ABC, Centralities comparison, *t*-test results

Adaptive ABC 3					**Adaptive ABC 3.b**					**Adaptive ABC 3.c**				
	0		**1**			**0**		**1**			**0**		**1**	
	p	*different*	*p*	*different*		*p*	*different*	*p*	*different*		*p*	*different*	*p*	*different*
0	-		-		**0**	-		-		**0**	-		-	
1	0.30621	F	-		**1**	0.38245	F	-		**1**	0.17363	F	-	
2	0.11774	F	0.09045	F	**2**	0.23515	F	0.26862	F	**2**	0.25100	F	0.23476	F

6 Conclusion

At the centre of interest of this thesis were the heuristics employed in solving the combinatorial optimisation problems, with the emphasis on scheduling problems. The core objective was to improve the performance of state-of-the art existing algorithms. This was attempted in several ways.

In the first part, deterministic heuristics were considered: NEH algorithm as one of the most popular constructive heuristics and the 2-opt algorithm as an improvement heuristic for permutative flowshop scheduling problem. The parallel implementation using the CUDA platform was developed for each of them.

Wide scale experimentation was performed for NEH solving permutative flowshop with makespan criterion, using the newly created data set based on the original Taillard data sets, which became a standard benchmark problem instances for the optimisation algorithms performance assessment, termed the extended Taillard data sets. The problem instances of size varying from 20 x 5, 20 x 10, 20 x 20, 50 x 5, 50 x 10, 50 x 20, 100 x 5, 100 x 10, 100 x 20, 200 x 10, 200 x 20 to 500 x 20 (the original Taillard data set) and 500 x 50, 700 x 20, 700 x 50 up to 1000 x 20 (the extended Taillard data set), 10 instances for each size, which makes up total of $16 \times 10 = 160$ unique instances, were evaluated by the original sequential and the newly developed parallel version of NEH, and the execution times compared. The statistical analysis of results using PRD as a measure has shown that the parallel implementation of NEH achieved the statistically significant speed-up over the sequential version for all instances greater than 50×10, with the best results of $6.18\times$ speed-up for the instance size 200×10, and the average speed-up of $2.62\times$. It can be therefore stated that there is a potential for speed-up.

Further development of this algorithm is two fold: the parallel implementation can be further optimised by fine-tuning the CUDA settings (launch configuration), as well as improved memory mapping using better suited cache types, supporting wider range of CUDA enabled GPU generations. The naive implementation of flow shop makespan evaluation could be also parallelised, achieving further acceleration. As the second aspect, the NEH heuristic would be used to provide the initial solution for the selected evolutionary algorithm (its CUDA based implementation), thus increasing the chance of finding better quality optima in addition to reducing the overall time required for the search.

The experimentation of somewhat reduced extent was performed for the 2-opt algorithm, solving again permutative FSS with makespan criterion, using only subset of standard Taillard data sets, the problem instances from 5×20 to 20×200, evaluating the three instances for each of the sizes, hence performing 33 experiments in total. Both the execution times and the optimal solutions found were compared between sequential and parallel version. No significant difference was proved between the quality of solutions, on the other hand, the execution time was again significantly reduced, starting from the instances of the size 100×5, with the best speed-up achieved for the instances size 200×10 and 200×20, yielding $1.92\times$ acceleration over the sequential version.

For the future goals, most of the same approaches as for the NEH can be employed. Further optimisation of the launch configuration and the improved memory mapping

could be used to achieve better results, together with the parallel flowshop evaluation. The broader support of compute capabilities could also be included. The accelerated version of 2-opt can then be also used to support the parallel implementation of an evolutionary algorithm, as it is the standard practice of embedding the local search into EA's to further improve the solutions generated by EA's search mechanisms.

In the second part, the effects of stochasticity on the selected combinatorial optimisation problems solving EA, the DABC algorithm, were explored. The standard PRNG, Mersenne Twister, was replaced by the PRNG based on the outputs of nine different chaotic piecewise linear one-dimensional maps. These nine unique chaos-based PRNGs then formed nine different modified versions of DABC; CDABC, whose performances were empirically assessed and compared against one another.

In the first stage of research, all of these algorithms performances were evaluated on the lot-streaming flowshop scheduling problem with setup times and no-idle constraint, using DABC with Mersenne Twister as a reference. Two unique CPRNG generated datasets, the Lozi data set using Lozi chaotic map, and the Dissipative data set using Dissipative chaotic map as CPRNG, were used for the experimentation. Each of them contained problem instances of 10 jobs $\times$ 5 machines, 20 jobs $\times$ 10 machines, 50 jobs $\times$ 25 machines, 75 jobs $\times$ 30 machines and 100 jobs $\times$ 50 machines, 5 instances of each size, making up the total of 25 unique instances in each of the two data sets. For both data sets, 15 repeated experiments were performed, forming the total of $2 \times 25 \times 15 = 750$ experiments for each algorithm variant, $10 \times 750 = 7500$ experiments in total. It was shown that the Tinkerbell, Delayed Logistic, Burgers and Lozi map yielded significantly better results than the original DABC with Mersenne Twister both for Lozi data sets and Dissipative data sets. To sum up the results, the five best performing PRNGs for Lozi data sets were: Tinkerbell, Delayed Logistic, Burgers, Lozi map and Mersenne Twister. For the Dissipative data sets, the ordering by performance was as follows: Tinkerbell, Delayed Logistic, Lozi, Burgers map and Mersenne Twister.

In the second stage, the CDABC variants were employed to solve the flowshop with no-wait constraint (zero-intermediate storage), using standard Taillard data set as a benchmark. For each of the 12 instance sizes, 10 instances per size, 14 repeated experiments were performed to obtain statistically valid sample, making the total of $12 \times 10 \times 14 = 1680$ experiments per variant, $10 \times 1680 = 16800$ experiments altogether. Again, statistically significant improvements were achieved. The order of the five best performing variants was as follows: Delayed Logistic, Tinkerbell, Burgers, Lozi, Mersenne Twister. This has shown that that the top four CDABC variants outperform Mersenne Twister on yet another problem, furthermore using the data sets generated using the standard PRNG as well, hence the previous results achieved in solving FSSLS instances generated by chaos-based PRNGs were not entirely problem and stochasticity specific.

Finally, the four best performing chaotic maps from previous experiments were used to solve the problems other than those in the manufacturing field, the capacitated vehicle routing problem and the quadratic assignment problem, solving the problem instances from the OR Library and the Taillard CVRP problem instances, respectively.

The QAP data set contains 17 problem instances. For each of them, 15 experiments

were performed, providing the total of $17 \times 15 = 255$ experiments per algorithm variant, $5 \times 255 = 1275$ experiments in total. CVRP range from 75 to 385 customers, consisting of 13 problem instances. For each of them, 15 experiments were performed, giving the total of $13 \times 15 = 195$ experiments per algorithm variant, $5 \times 195 = 975$ experiments in total.

In both cases the statistically significant improvement was obtained. The order of the best performing algorithms is as follows: Tinkerbell, Delayed Logistic, Burgers map, Lozi map, Mersenne Twister for QAP problem instances; Tinkerbell, Burgers map, Delayed Logistic, Lozi map, Mersenne Twister for the CVRP problems. This comes at a cost of slightly decreased speed of CDABC variants, compared to the DABC with Mersenne Twister.

The future development plans include the exploration of different types of chaotic systems used as the CPRNGs, for instance the coupled maps, as well as further applications of CPRNGs in solving difficult combinatorial optimisation problems, since the superiority was empirically proven.

For the last part of the thesis, the complex networks properties, namely vertex centrality measures were used to modify the ABC algorithm, providing the basis of self-adaptive mechanism, which replaces the solutions that don't contribute to the population development with the new ones. Three different centralities: weighted degree, closeness, betweenness were implemented inside five different approaches to the design of such algorithm: Adaptive ABC 1, Adaptive ABC 2, Adaptive ABC 3, Adaptive ABC 3.b and Adaptive ABC 3.c. Both Adaptive ABC 1 and 2 were implemented using all three centrality measures - Adaptive ABC 1 splitting the population conceptually into three parts for the purpose of vertex centrality evaluation only, Adaptive ABC 2 using ensemble approach creating three separated populations, each with its own network and assigned centrality measure. Adaptive ABC 3 used centrality measure type provided as parameter, enabling the assessment of the effect of different centrality measures. This was extended by incorporation of elitism, always keeping certain ratio of best quality solutions in the population, in the algorithms Adaptive ABC 3.b and Adaptive ABC 3.c.

The functions of the standard test set for continuous optimisation were used for experimentation: Schwefel, De Jong 1, De Jong 3, De Jong 4, Rosenbrock's Saddle, Rastrigin, Griewangk, Sine Envelope Sine Wave, Ackley One, Ackley Two, Egg Holder, Michalewicz, Master's Cosine Wave, and Schekel's Foxhole, with the dimensions of 10, 20, 30 for the first experiments set; the same functions plus Stretch V Sine Wave for the second experiments set with the dimensions of 10, 20, 30, 40, 50, 75, 100.

The first experiment set was conducted to compare the ABC, Adaptive ABC 1 and Adaptive ABC 2. Thirty experiments were performed for each problem and dimension, trying the different number of solutions in the population: 15, 30, 45 and 60. This makes up together 14 problems $\times$ 3 dimensions $\times$ 4 number of solutions settings $\times$ 30 $=$ 5040 experiments for each algorithm, $3 \times 5040 = 15120$ experiments in total.

The second experiment set was extended with the Adaptive ABC 3, Adaptive ABC 3.b and Adaptive ABC 3.c. All five algorithms were compared against the standard ABC. Fifteen repetitions were done for each problem and each dimension, resulting in 15 problems $\times$ 7 dimensions $\times$ 15 $=$ 1575 experiments for each algorithm. Another experimen-

tation on this set was conducted to compare the effects of different centrality measures against one another, using the same setting described above for degree, closeness and betweenness, in each of Adaptive ABC 3, Adaptive ABC 3.b and Adaptive ABC 3.c, comparing the results separately. This makes up $1575 \times 3 = 4725$ experiments for each of the latter algorithms, hence 14175 experiments in total.

For the first experiment set, the Adaptive ABC 2 tends to be better for the problems with 30 variables, reaching the best values (in comparison with both ABC and Adaptive ABC 1) in 7 out of 14 problems. On the other hand, the Adaptive ABC 1 tends to be better than Adaptive ABC 2 for the dimensions 10 and 20, however, the results are comparable with the standard ABC algorithm in all cases.

In the second experiment set, where the parameters were tuned, the statistically significant difference was proven for all of the six tested algorithms pairwise, except for the Adaptive ABC 1 and Adaptive ABC 3. The order of the algorithms sorted by the average performances is following: Adaptive ABC 3.b has the best overall average. The 2nd best average was achieved by the Adaptive ABC 3.c. The standard ABC follows with the Adaptive ABC 3, Adaptive ABC 1 and Adaptive ABC 2 having the worst overall average. Comparing by the individual problems separately, The Adaptive ABC 3.b is better than the original ABC in ten out of total 15 problems. The Adaptive ABC 3.c is better in nine problems, Adaptive ABC 3 in three problems. The Adaptive ABC 1 improves upon ABC in only two of the problems, whereas Adaptive ABC 2 gives four better results than ABC. It can be therefore concluded, that the versions with elitism perform significantly better on average, and that in accordance with no free lunch theorem, each of the algorithms can be potentially useful at least in subset of problem classes. The comparison of measure types didn't find any significant differences, the performance of all of the three centralities was similar.

For the future path, since the selected centralities performed similarly well for all of the algorithm variants, it would be sufficient to retain only one of them. Different modifications to the existing algorithms could be produced, incorporating more sophisticated complex network properties and different centrality measures (Katz Centrality for example) to asses the nodes influence in the network. It is obviously necessary to incorporate the elitism into such algorithms, in order to retain the good quality solutions in the population and not to randomize the search to a large extent. Different schema of the least influential solutions replacement could also be used, for instance creating the new solutions in the proximity of the already found good ones, thus accelerating the convergence. Finally, this approach would be implemented in a discrete evolutionary algorithm which also forms complex networks through the course of its iterations, to support solving of the combinatorial optimisation problems.

7 References

[1] B. Alatas, E. Akin, and A.B. Ozer. Chaos embedded particle swarm optimization algorithms. *Chaos, Solitons and Fractals*, 40(4):1715–1734, 2009.

[2] K. Alligood, T. Sauer, and J. Yorke. *Chaos*. Springer, Germany, 1997.

[3] D.G. Aronson, M.A. Chory, G.R. Hall, and R. McGehee. A discrete dynamical system with subtly wild behavior. In D. Davendra, editor, *New Approaches to Nonlinear Problems in Dynamics*, pages 339–359. SIAM Publications, Philadelphia, Pennsylvania, 1980.

[4] A. Bagheri, M. Zandieh, Iraj Mahdavi, and M. Yazdani. An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems-The International Journal of Grid Computing and Escience*, 26(4):533–541, APR 2010.

[5] Kenneth R. Baker and Dan Trietsch, editors. *Principles of Sequencing and Scheduling*. Wiley, USA, 2009.

[6] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351 – 385, 2008.

[7] R. Baldacci, P. Toth, and D. Vigo. Exact algorithms for routing problems under vehicle capacity constraints. *Annals of Operations Research*, 175(1):213 – 245, 2010.

[8] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Science*, 101:3747–3752, March 2004.

[9] J. E. Beasley. Operations research library, 2014. `http://www.brunel.ac.uk/~mastjjb/jeb/info.html`.

[10] J.M. Burgers. Mathematical examples illustrating relations occurring in the theory of turbulent fluid motion. In F.T.M. Nieuwstadt and J.A. Steketee, editors, *Selected Papers of J. M. Burgers*, pages 281–334. Springer Netherlands, 1995.

[11] Jen Huei Chang and Huan Neng Chiu. A comprehensive review of lot streaming. *International Journal of Production Research*, 43(8):1515–1536, 2005.

[12] L.-Y. Chuang, S.-W. Tsai, and C.-H. Yang. Chaotic catfish particle swarm optimization for solving global numerical optimization problems. *Applied Mathematics and Computation*, 217(16):6900–6916, 2011.

[13] L.D.S. Coelho, T.C. Bora, and L. Lebensztajn. A chaotic approach of differential evolution optimization applied to loudspeaker design problem. *IEEE Transactions on Magnetics*, 48(2):751–754, 2012.

[14] D. Connolly. An improved annealing scheme for the qap. *European Journal of Operation Research*, 46:93 – 100, 1990.

[15] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[16] M. Czapiński and S. Barnes. Tabu search with two approaches to parallel flowshop evaluation on cuda platform. *Journal of Parallel and Distributed Computing*, 71(6):802–811, 2011.

[17] Davendra D. and Zelinka I. Flow shop scheduling using self organising migrating algorithm. In *Proc. 22nd European Conference of Modelling and Simulation*, pages 195–200, Nisosia, Cyprus, June 2008.

[18] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80 – 91, 1959.

[19] George B. Dantzig. Origins of the simplex method. In *Technical Report SOL 87-5*, Standford University, May 1987.

[20] D. Davendra, M. Bialic-Davendra, and R. Senkerik. Scheduling the lot-streaming flowshop scheduling problem with setup time with the chaos-induced enhanced differential evolution. *Proceedings of the 2013 IEEE Symposium on Differential Evolution, SDE 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013*, pages 119–126, 2013.

[21] Donald Davendra. Flowshop lot-streaming problem data sets, 2012.

[22] Donald Davendra. Taillard extended data sets, 2014.

[23] Donald Davendra, Roman Senkerik, Ivan Zelinka, Michal Pluhacek, and Magdalena Bialic-Davendra. Utilising the chaos-induced discrete self organising migrating algorithm to solve the lot-streaming flowshop scheduling problem with setup time. *Soft Computing*, 18(4):669–681, 2014.

[24] Donald Davendra, Ivan Zelinka, Magdalena Bialic-Davendra, Roman Senkerik, and Roman Jasek. Discrete self-organising migrating algorithm for flow-shop scheduling with no-wait makespan. *Mathematical and Computer Modelling*, 57(1-2):100–110, JAN 2013.

[25] Donald Davendra, Ivan Zelinka, and Roman Senkerik. Chaos driven evolutionary algorithms for the task of pid control. *Computers & Mathematics with Applications*, 60(4):1088 – 1104, 2010.

[26] Donald Davendra, Ivan Zelinka, Roman Senkerik, and Michal Pluhacek. Complex network analysis of discrete self-organising migrating algorithm. In Ivan Zelinka, Ponnuthurai Nagaratnam Suganthan, Guanrong Chen, Vaclav Snasel, Ajith Abraham, and Otto Rössler, editors, *Nostradamus 2014: Prediction, Modeling and Analysis of Complex Systems*, volume 289 of *Advances in Intelligent Systems and Computing*, pages 161–174. Springer International Publishing, 2014.

[27] Donald Davendra, Ivan Zelinka, Roman Senkerik, and Michal Pluhacek. Complex network analysis of evolutionary algorithms applied to combinatorial optimisation problem. In Pavel Krömer, Ajith Abraham, and Václav Snášel, editors, *Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014*, volume 303 of *Advances in Intelligent Systems and Computing*, pages 141–150. Springer International Publishing, 2014.

[28] Ulrich Derigs. *Optimization and operations research*. Eolss Publishers Co Ltd, Oxford, 2009.

[29] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, Nov 2006.

[30] M. Eslami, H. Shareef, and A. Mohamed. Power system stabilizer design using hybrid multi-objective particle swarm optimization with chaos. *Journal of Central South University of Technology (English Edition)*, 18(5):1579–1588, 2011.

[31] Iztok Fister, Iztok Fister Jr., Xin-She Yang, and Janez Brest. A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*, 13(0):34 – 46, 2013.

[32] C. Fleurent and J. Ferland. Genetic hybrids for the quadratic assignment problem. *Operations Research Quarterly*, 28:167 – 179, 1994.

[33] Onwubolu G. and Davendra D. *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*. Springer, Germany, 2009.

[34] A.H. Gandomi, G.J Yun, X.-S Yang, and S Talatahari. Chaos-enhanced accelerated particle swarm optimization. *Communications in Nonlinear Science and Numerical Simulation*, 18(2):327–340, 2013.

[35] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[36] B. L. Golden, S. Raghavan, and E. A. Wasil. *The vehicle routing problem: latest advances and new challenges*. Society for Industrial and Applied Mathematics,, Springer, Germany, 2008.

[37] J. Grabowski and J. Pempera. Sequencing of jobs in some production system. *European Journal of Operational Research*, pages 535–550, 2000.

[38] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer, 2002.

[39] N. Hall and C. Sriskandarayah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, pages 510–525, 1996.

[40] Yu-Yan Han, J. J. Liang, Quan-Ke Pan, Jun-Qing Li, Hong-Yan Sang, and N. N. Cao. Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. *International Journal of Advanced Manufacturing Technology*, 67(1-4, SI):397–414, JUL 2013.

[41] Charles Herring and Palmore Julian. Random number generators are chaotic. *ACM SIGPLAN*, 11:1–4, 1989.

[42] W.-C. Hong, Y Dong, L.-Y. Chen, and S.-Y Wei. Svr with hybrid chaotic genetic algorithms for tourism demand forecasting. *Applied Soft Computing Journal*, 11(2):1881–1890, 2011.

[43] Sang Hongyan, Gao Liang, and Pan Quanke. Discrete Artificial Bee Colony Algorithm for Lot-streaming Flowshop with Total Flowtime Minimization. *Chinese Journal of Mechanical Engineering*, 25(5):990–1000, SEP 2012.

[44] H. Jiang, C.K. Kwong, Z. Chen, and Y.C. Ysim. Chaos particle swarm optimization and t-s fuzzy modeling approaches to constrained predictive control. *Expert Systems with Applications*, 39(1):194–201, 2012.

[45] David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997.

[46] P.J. Kalczynski and J. Kamburowski. On the neh heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35(1):53–60, 2007.

[47] D. Karaboga and B. Basturk. On the performance of artificial bee colony (ABC) algorithm. *APPLIED SOFT COMPUTING*, 8(1):687–697, JAN 2008.

[48] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471, NOV 2007.

[49] Dervis Karaboga and Bahriye Basturk. Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In Patricia Melin, Oscar Castillo, LuisT. Aguilar, Janusz Kacprzyk, and Witold Pedrycz, editors, *Foundations of Fuzzy Logic and Soft Computing*, volume 4529 of *Lecture Notes in Computer Science*, pages 789–798. Springer Berlin Heidelberg, 2007.

[50] A. Kazem, E. Sharifi, F.K. Hussain, M. Saberi, and O.K. Hussain. Support vector regression with chaos-based firefly algorithm for stock market price forecasting. *Applied Soft Computing Journal*, 13(2):947–958, 2013.

[51] J Kennedy and R Eberhart. Particle swarm optimization. In *1995 IEEE International Conference on Neural Networks Proceedings, Vols 1-6*, pages 1942–1948, 1995.

[52] B. Kim, J. Shim, and M. Zhang. Comparison of tsp algorithms., 1998. Project for Facilities Planning and Materials Handling.

[53] David B Kirk and W Hwu Wen-mei. *Programming massively parallel processors: a hands-on approach*. Newnes, 2012.

[54] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Tabu search with two approaches to parallel flowshop evaluation on cuda platform. *Optimization by Simulated Annealing*, 220:671–680, 1983.

[55] T. Koopans and M. Beckman. Assignment problems and the location of economic activities. *Econometrica*, 25:53 – 76, 1957.

[56] Media Research Lab. Media research lab, 2014. `http://mrl.cs.vsb.cz`.

[57] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345 – 358, 1992.

[58] D Lehmer. Mathematical methods in large-scale computing units. *Ann. Computing Lab, Harvard University*, 26:141–146, 1951.

[59] J. Li, L. Yang, J.-L. Liu, D.-L. Yang, and C. Zhang. Multi-objective reactive power optimization based on adaptive chaos particle swarm optimization algorithm. *Dianli Xitong Baohu yu Kongzhi/Power System Protection and Control*, 39(9):26–31, 2011.

[60] Jun-Qing Li, Quan-Ke Pan, and Kai-Zhou Gao. Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 55(9-12):1159–1169, AUG 2011.

[61] W. Liang, L. Zhang, and M. Wang. The chaos differential evolution optimization algorithm and its application to support vector regression machine. *Journal of Software*, 6(7):1297–1304, 2011.

[62] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[63] Shih-Wei Lin, Zne-Jung Lee, Kuo-Ching Ying, and Chou-Yuan Lee. Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, 36(2, Part 1):1505 – 1512, 2009.

[64] F. Liu, H. Duan, and Y. Deng. A chaotic quantum-behaved particle swarm optimization based on lateral inhibition for image matching. *Optik*, 123(21):1955–1960, 2012.

[65] Rene Lozi. New enhanced chaotic number generators. *Indian Journal of Industrial and Applied Mathematics*, 1(1):1–23, 2008.

[66] Rene Lozi. Chaotic pseudo random number generators via ultra weak coupling of chaotic maps and double threshold sampling sequences. In *ICCSA 2009 The 3rd International Conference on Complex Systems and Applications*, pages 1–5, University of Le Havre, France, June 2009.

[67] H. Lu, R. Niu, J. Liu, and Z. Zhu. A chaotic non-dominated sorting genetic algorithm for the multi-objective automatic test task scheduling problem. *Applied Soft Computing Journal*, 13(5):2790–2802, 2013.

[68] V.C. Mariani, A.R.K. Duck, F.A. Guerra, L.D.S. Coelho, and R.V. Rao. A chaotic quantum-behaved particle swarm approach applied to optimization of heat exchangers. *Applied Thermal Engineering*, 42:119–128, 2012.

[69] M. Matsumoto. Mersenne twister webpage, 2012. `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/earticles.html`.

[70] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transaction on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[71] M. Metlicka and D. Davendra. Chaos-driven discrete artificial bee colony. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2947–2954, July 2014.

[72] M. Metlicka and D. Davendra. Ensemble centralities based adaptive artificial bee algorithm. In *Evolutionary Computation (CEC), 2015 IEEE Congres, Sendai, Japan. (accepted)*, 2015.

[73] M. Metlicka, D. Davendra, F. Hermann, M. Meier, and M. Amann. Gpu accelerated neh algorithm. In *Computational Intelligence in Production and Logistics Systems (CIPLS), 2014 IEEE Symposium on*, pages 114–119, Dec 2014.

[74] Magdalena Metlická and Donald Davendra. Scheduling the flowshop with zero intermediate storage using chaotic discrete artificial bee algorithm. In Ivan Zelinka, Ponnuthurai Nagaratnam Suganthan, Guanrong Chen, Vaclav Snasel, Ajith Abraham, and Otto RĂ¶ssler, editors, *Nostradamus 2014: Prediction, Modeling and Analysis of Complex Systems*, volume 289 of *Advances in Intelligent Systems and Computing*, pages 141–152. Springer International Publishing, 2014.

[75] Magdalena Metlicka and Donald Davendra. Chaos driven discrete artificial bee algorithm for location and assignment optimisation problems. *Swarm and Evolutionary Computation*, (0):–, 2015.

[76] M. Nawaz, E.E. Enscore Jr., and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[77] NVIDIA. Kepler gk110, 2012.

[78] NVIDIA. Cuda c programming guide, February 2014.

[79] NVIDIA. Cuda c best practices guide. online, 2015.

[80] Godfrey Onwubolu and Donald Davendra. Differential evolution for permutation – based combinatorial problems. In GodfreyC. Onwubolu and Donald Davendra, editors, *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, volume 175 of *Studies in Computational Intelligence*, pages 13–34. Springer Berlin Heidelberg, 2009.

[81] Ahmet Bedri Ozer. Cide: Chaotically initialized differential evolution. *Expert Systems with Applications*, 37(6):4632 – 4641, 2010.

[82] J Palmore and J McCauley. Shadowing by computable chaotic orbits. *Physics Letters A*, 121:399, 1987.

[83] Quan-Ke Pan, M. Fatih Tasgetiren, P. N. Suganthan, and T. J. Chua. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Inf. Sci.*, 181(12):2455–2468, June 2011.

[84] Quan-Ke Pan and RubĂ©n Ruiz. An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *Omega*, 40(2):166–180, April 2012.

[85] Quan-Ke Pan, M. Fatih Tasgetiren, P. N. Suganthan, and T. J. Chua. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information Sciences*, 181(12):2455–2468, JUN 15 2011.

[86] Quan-Ke Pan, Ling Wang, and Liang Gao. A chaotic harmony search algorithm for the flow shop scheduling problem with limited buffers. *Applied Soft Computing*, 11(8):5270 – 5280, 2011.

[87] C. Peng, H. Sun, J. Guo, and G. Liu. Dynamic economic dispatch for wind-thermal power system using a novel bi-population chaotic differential evolution algorithm. *International Journal of Electrical Power and Energy Systems*, 42(1):119–126, 2012.

[88] M. Pinedo. *Scheduling: theory, algorithms and systems*. Prentice Hall, Inc., New Jersey, 1995.

[89] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[90] M. Pluhacek, R. Senkerik, D. Davendra, Z. Kominkova Oplatkova, and I. Zelinka. On the behavior and performance of chaos driven pso algorithm with inertia weight. *Computers and Mathematics with Applications*, 66(2):122–134, 2013.

[91] M. Pluhacek, R. Senkerik, D. Davendra, and I. Zelinka. Pid controller design for 4th order system by means of enhanced pso algorithm with lozi chaotic map. In *Proceedings of the 18th International Conference on Soft Computing, MENDEL*, pages 35–39, 2012.

[92] M. Pluhacek, R. Senkerik, I. Zelinka, and D. Davendra. Chaos pso algorithm driven alternately by two different chaotic maps-an initial study. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pages 2444–2449, 2013.

[93] Reeves C. R. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[94] W. Raaymakers and J. Hoogeveen. Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, pages 131–151, 2000.

[95] Shahriar Farahmand Rad, Rubén Ruiz, and Naser Boroojerdian. New high performing heuristics for minimizing makespan in permutation flowshops. *Omega*, 37(2):331 – 345, 2009.

[96] C. Rajendran. A no-wait flowshop scheduling heuristic to minimize makespan. *Journal of the Operational Research Society*, pages 472–478, 1994.

[97] J. Sanders and E. Kandrot. *CUDA by example*. Addison-Wesley, 1st print. edition, 2010.

[98] Subhash C. Sarin and Purneet Jaiprakash. *Flow Shop Lot Streaming*. Springer, Berlin, 2007.

[99] M.W.P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75 – 85, 1990.

[100] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.

[101] R. Senkerik, D. Davendra, I. Zelinka, Z. Oplatkova, and M Pluhacek. Optimization of the batch reactor by means of chaos driven differential evolution. *Advances in Intelligent Systems and Computing*, 188 AISC:93–102, 2013.

[102] R. Senkerik, M. Pluhacek, D. Davendra, I. Zelinka, and Z. Kominkova Oplatkova. Chaos driven evolutionary algorithm: A new approach for evolutionary optimization. *International Journal of Mathematics and Computers in Simulation*, 7(4):363–368, 2013.

[103] Roman Senkerik, Ivan Zelinka, Michal Pluhacek, Donald Davendra, and Zuzana Oplatkova Kominkova. Chaos enhanced differential evolution in the task of evolutionary control of selected set of discrete chaotic systems. *The Scientific World Journal*, 2014, 2014.

[104] Lin Shen and Kernighan B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[105] J. Sprott. *Chaos and Time-Series Analysis*. Oxford University Press, UK, 2003.

[106] Toni Stojanovski and Ljupco Kocarev. Chaos-based random number generators, part i: Analysis. *IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications*, 48(3):281–288, 2001.

[107] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443 – 455, 1991.

[108] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.

[109] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.

[110] E. Taillard. Vehicle routing problem instances, 2012. `http://mistic.heig-vd.ch/taillard/`.

[111] E. Taillard. Taillard flowshop data sets, 2014.

[112] M. Fatih Tasgetiren, Yun-Chia Liang, Mehmet Sevkli, and Gunes Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3):1930 – 1947, 2007.

[113] M. Fatih Tasgetiren, Quan-Ke Pan, P. N. Suganthan, and Angela H-L Chen. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*, 181(16):3459–3475, AUG 15 2011.

[114] M. Fatih Tasgetiren, Quan-Ke Pan, P.N. Suganthan, and Adalet Oner. A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Applied Mathematical Modelling*, 37:6758 – 6799, 2013.

[115] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics,, Philadelphia, USA, 2001.

[116] Remco van der Hofstad. Random graphs and complex networks. vol. i. online, 2014.

[117] Jiaoe Wang, Huihui Mo, Fahui Wang, and Fengjun Jin. Exploring the network structure and nodal centrality of china's air transport network: A complex network approach. *Journal of Transport Geography*, 19(4):712 – 721, 2011.

[118] R.-Q. Wang, C.-H. Zhang, and K. Li. Multi-objective genetic algorithm based on improved chaotic optimization. *Kongzhi yu Juece/Control and Decision*, 26(9):1391–1397, 2011.

[119] W.-B. Wang, Q.-Y. Feng, and D. Liu. Application of chaotic particle swarm optimization algorithm to pattern synthesis of antenna arrays. *Progress in Electromagnetics Research*, 115:173–189, 2011.

[120] R.R. Whitehead and N. MacDonald. A chaotic mapping that displays its own homoclinic structure. *Physica D: Nonlinear Phenomena*, 13(3):401 – 407, 1984.

[121] Darrell Whitley, Soraya Rana, John Dzubera, and Keith E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245 – 276, 1996.

[122] Q. Wu. A self-adaptive embedded chaotic particle swarm optimization for parameters selection of wv-svm. *Expert Systems with Applications*, 38(1):184–192, 2011.

[123] X.-B. Xu, K.-F. Zheng, D. Li, B. Wu, and Y.-X. Yang. New chaos-particle swarm optimization algorithm. *Tongxin Xuebao/Journal on Communications*, 33(1):24–30+37, 2012.

[124] Xin-She Yang. Chapter 10 - bat algorithms. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 141 – 154. Elsevier, Oxford, 2014.

[125] Y. Yang, Y. Wang, X. Yuan, and F. Yin. Hybrid chaos optimization algorithm with artificial emotion. *Applied Mathematics and Computation*, 218(11):6585–6611, 2012.

[126] Xiaohui Yuan, Bo Cao, Bo Yang, and Yanbin Yuan. Hydrothermal scheduling using chaotic hybrid differential evolution. *Energy Conversion and Management*, 49(12):3627 – 3633, 2008.

[127] I. Zelinka, M. Chadli, D. Davendra, R. Senkerik, M. Pluhacek, and J. Lampinen. Do evolutionary algorithms indeed require random numbers? extended study. *Advances in Intelligent Systems and Computing*, 210:61–75, 2013.

[128] I. Zelinka, M. Chadli, D. Davendra, R. Senkerik, M. Pluhacek, and J. Lampinen. Hidden periodicity - chaos dependance on numerical precision. *Advances in Intelligent Systems and Computing*, 210:47–59, 2013.

[129] Ivan Zelinka. Soma – self-organizing migrating algorithm. In *New Optimization Techniques in Engineering*, volume 141 of *Studies in Fuzziness and Soft Computing*, pages 167–217. Springer Berlin Heidelberg, 2004.

[130] Ivan Zelinka, Zuzana Oplatková, Miloš Šeda, Pavel Ošmera, and František Včelař. *Evoluční výpočetní techniky - principy a aplikace*. BEN, Praha, 1. české vyd. edition, 2009.

[131] M. Zhang and G. Li. Network intrusion detection based on least squares support vector machine and chaos particle swarm optimization algorithm. *Journal of Convergence Information Technology*, 7(4):169–174, 2012.

A CD with experiment data and source codes

The accompanying CD contains the electronic form of this document, experiment outputs and the source codes with short manuals for each of the programs. The structure of CD is following:

- **Code** :

  Source codes of each of the programs in separate folder.

- **ExperimentData** :

  Experiment outputs in separate folders.

- **Manuals** :

  Manuals, readmes and examples of usage.

- **Text** :

  Electronic form of this thesis, *thesis.pdf*.