A Study of Particle Swarm Optimization Trajectories for Real-Time Scheduling

by

Dario Schor

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

in partial fulfilment of the requirements of the degree of

Master of Science

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Canada

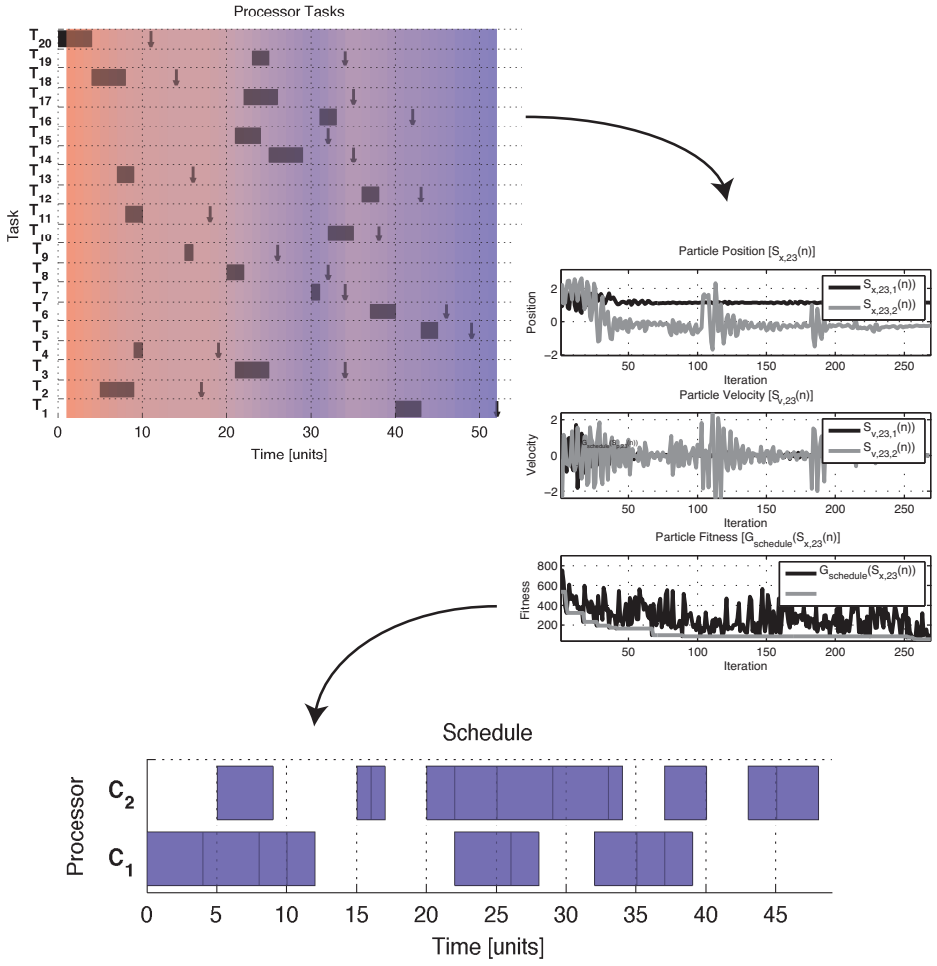
Abstract

Scheduling of aperiodic and independent tasks in hard real-time symmetric multiprocessing systems is an NP-complete problem that is often solved using heuristics like particle swarm optimization (PSO). The performance of these class of heuristics, known as evolutionary algorithms, are often evaluated based on the number of iterations it takes to find a solution. Such metrics provide limited information on how the algorithm reaches a solution and how the process could be accelerated.

This thesis presents a methodology to analyze the trajectory formed by candidate solutions in order to analyze them in both the time and frequency domains at a single scale. The analysis entails (i) the impact of different parameters for the PSO algorithm, and (ii) the evolutionary processes in the swarm. The work reveals that particles have a directed movement towards a solution during a transient phase, and then enter a steady state where they perform an unguided local search.

The scheduling algorithm presented in this thesis uses a variation of the minimum total tardiness with cumulative penalties cost function, that can be extended to suit different system needs. The experimental results show that the scheduler is able to distribute tasks to meet the real-time deadlines over 1, 2, and 4 processors and up to 30 tasks with overall system loads of up to 50% in fewer than 1,000 iterations. When scheduling greater loads, the scheduler reaches local solutions with 1 to 2 missed deadlines, while larger tasks sets take longer to converge. The trajectories of the particles during the scheduling algorithm are examined as a means to emphasize the impact of the behaviour on the application performance and give insight into ways to improve the algorithm for both space and terrestrial applications.

Visual Abstract



Acknowledgements

There were many people who supported and encouraged me throughout this degree without whom I would not have completed this work.

I would like to thank my advisor Dr. Witold Kinsner for challenging me on a daily basis and always raising expectations. Furthermore, thank you for encouraging me to do more than just the thesis. This experience would not have been the same without UMSATS, UMARS, IEEE, and many other activities.

There are many people within the Electrical and Computer Engineering Department that helped me throughout this process. In particular I'd like to single out Dr. Ken Ferens and Dr. Bob McLeod for their encouragement throughout. Ken Biegun, Mount-first Ng, and Amy Dario for all their help not just in the thesis, but in all the random projects I found myself working in.

Special thanks to Prof. Ron Britton, Mr. Malcolm Symonds, Mr. Allan McKay, Mr. Alan Thoren, and Mr. Gord Klimenko with whom I shared many conversations throughout the years that spanned research, engineering, and life in general. These discussions served as motivation throughout the thesis and other activities alike.

Special thanks to members of the Delta Research Group: Michael Potter, Lily Woo, David Terrazas, Greg Linton, Mohamed Nasri, and Kathryn Marcynuk with whom I learned, discussed, and shared ideas on a regular basis.

Thank you to current and former members of Magellan Aerospace Winnipeg that supported and collaborated in various parts of this thesis: Neil Gadhok, Doug Cornelsen, Peter Wtorek, Darrin Gates, Diane Kotelko, Philip Ferguson, and Walter Czyrnyj. Furthermore, thank you to Joel Sherrill from RTEMS and Gedare Bloom for their input in the design of the scheduler algorithm.

I would like to thank my family and friends for their support throughout the years. Thank you to my parents Ana and Carlos for always encouraging me to pursue my passions. Special thank you to my sister Maia for challenging everything I say, do, and think.

Finally, but certainly not last, thank you to my friends for their motivation and words of encouragement. In particular, special thanks to Matial Wengiel, Ariel Brawerman, Sari Levi, Matthew Leibl, Paula Sturrey, Tyler Loewen, Sam Kovnats, Steve Woodrow, Jane Polak Scowcroft, Mark Roy, Travis Friesen, Matthew Sebastian, Kane Anderson, Troy Denton, Ahmad Byagowi, Matthew Woelk, and Brady Russell.

This work was funded in part by Mitacs Accelerate through Magellan Aerospace Winnipeg and through the University of Manitoba Graduate Students Association Award.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

Notation: Scalars are denoted by plain text, italics. Vectors are denoted by bold text.

ℓBest	*local best*	63–67, 113, 140
gBest	*global best*	63, 64, 67, 113, 123, 140
AAC	*asymptotic algorithm complexity*	14, 15
ACO	*ant colony optimization*	37
ADC	*attitude determination and control*	2, 5
AMP	*asymmetric multiprocessing*	17–20, 28, 110
CASSIOPE	*CAscade, SmallSat and IOnospheric Polar Explorer*	4
CDH	*command and data handling*	2–5
CSA	*Canadian Space Agency*	4
DM	*deadline-monotonic*	29
DMA	*direct memory access*	17
EA	*evolutionary algorithms*	3, 7–9, 33, 54, 70–73, 75, 76, 79, 80, 98, 100, 105, 144–146
EDF	*early deadline first*	29–31
FIFO	*first-in-first-out*	28

FPGA	*field programmable gate array*	2, 5
FSW	*flight software*	4
GA	*genetic algorithm*	36–39, 54, 72, 76
GCD	*greatest common divisor*	27
IO	*input/output*	17, 18
LEO	*low-earth-orbit*	3, 4, 10
LLF	*least-laxity first*	29
LLREF	*largest local remaining execution time first*	14
LRETL	*local remaining execution time and local execution time*	14
MAW	*Magellan Aerospace Winnipeg*	2–5, 7, 128, 129
MSE	*mean square error*	85, 88, 144, A12
MTT	*minimum-total-tardiness*	103, 105, 110, 145, 147
NFL	*no-free-lunch*	75, 80
NPC	*NP-complete*	12, 14
ORBITALS	*Outer Radiation Belt Injection Transport and Loss Satellite*	4
PCB	*printed-circuit board*	2
pdf	*probability density function*	38
pmf	*probability mass function*	38

PSO	*particle swarm optimization*	1, 5–11, 34, 39–47, 49, 51, 53–56, 58, 61, 63–65, 68, 69, 72, 74, 76–78, 80, 82, 85, 89, 98, 100, 103, 106, 111–117, 120–122, 130, 140, 142–148, B1
RAM	*random access memory*	74
RCM	*Radarsat Constellation Mission*	4, 128–130
RK	*random-keys*	100–104, 107, 118, 122, 124, 140, 145, 146, 148, B2
RM	*rate-monotonic*	29, 98
RTOS	*real-time operating system*	2–5
SA	*simulated annealing*	34–39, 47, 58, 69, 72, 76, 100, 118
SMP	*symmetric multiprocessing*	5–7, 10, 11, 17, 18, 20, 22, 26, 28, 70, 76, 98, 110, 111, 126, 143, 145, 146
SSS	*strict-sense stationary*	89
SW	*small-world*	66
TSat	*triple pico-satellite*	xii, 98, 99
UMSATS	*University of Manitoba Space Applications and Technology Society*	98

VFDT	*variance fractal dimension trajectory*	147
WSS	*wide-sense stationary*	89, 92, 115, 116, 136, A2

List of Symbols

Notation: Scalars are denoted by plain text, italics. Vectors are denoted by bold text.

$\mu_{x,k,d}(n)$	Mean position of a particle ensemble during the n^{th} iteration. sort	85, 86, 88, A18
p_index	Index of the processor where a task is assigned. sort	105, 107
C_j	Refers to j^{th} processor in a multiprocessor system.	15, 17, 22
C_N	Number of processors in the system.	101, 106, 123
D	Number of dimensions in a cost function.	39, 43, 45, 123
d	Index of current dimension in a cost function.	48
ΔD	Distance between the current and the candidate solution in SA	35
d_index	Index of the dimension/task being allocated in the scheduling algorithm.	105
$G(\bullet)$	Cost function used in optimization algorithm.	47
$G_1(\mathbf{x})$	Sphere function given by Eq. 4.1	xii, xiii, 77, 83, 86, 88, 89, 93, 95, 97, A5
$G_2(\mathbf{x})$	Rosenbrock function given by Eq. 4.2	xiii, 77, 83, A12
$G_3(\mathbf{x})$	Rastrigin function given by Eq. 4.3	xii, xiii, 77, 84, 87, 88, 94, 96, 97, A21

$G_4(\mathbf{x})$	Griewank function given by Eq. 4.4	xiii, 77, 84, A28
n	Current iteration in an evolutionary algorithm (n^{th} iteration).	101
N_{max}	Maximum number of iterations in an evolutionary algorithm.	49, 51, 73, 113, 117, 123, 124
N_{runs}	Number of runs of an optimization algorithm.	82, 86
Q_α	Convergence rate metric for an EA.	71, 72
Q_β	Degree of exploration metric for an EA.	71, 72
Q_γ	Storage and system size metric for an EA.	71, 73
Q_δ	Adaptability metric for an EA.	71, 74
Q_ϵ	Multi-scale capabilities metric for an EA.	71, 75
$R_A(t)$	Schedulability test for aperiodic, single-instance tasks. Also serves as a measure of surplus computing power.	26, 119, 129, 130, 147
R_{P3}	Schedulability test for periodic tasks.	27
S_χ	Constriction coefficient in PSO algorithm.	59, 60
$\mathbf{S}_{fp}$	Cached fitness of particle's best position in PSO.	48
$\mathbf{S}_{fx}$	Cached fitness of particle's current position in PSO.	48
S_g	Index of the best particle in the neighbourhood for a given particle.	43, 49, 51, 58, 63, 67, 113
S_{gnext}	Index of the best particle in the neighbourhood for a given particle to be used in the next iteration.	51
S_K	Number of particles in PSO	39, 46, 47, 51, 56, 57, 64, 113, 123
S_k	Particle index count in PSO	xii, 64, 101, 113, 115
S_n	Size of particle's neighbourhood.	49, 63, 66, 67, 113
$\mathbf{S}_p$	Past position for one particle in PSO.	43, 45–48

$S_{\varphi 1}$	Personal influence weight in PSO.	43, 44, 61, 62, 81, 93, 112, 123
$S_{\varphi 2}$	Social influence weight in PSO.	43, 44, 58, 61, 62, 81, 93, 112, 123
$\mathbf{S}_v$	Velocity for one particle in PSO.	43, 45, 88
S_{Vmax}	Maximum velocity in PSO algorithm.	58–61, 75, 114, 123, 128
S_{Vmin}	Minimum velocity in PSO algorithm.	58, 123
S_{ω}	Inertia weight in PSO algorithm.	58–60, 62, 112, 114, 123
$S_{\omega,eph}$	Duration of gradient S_{ω} in PSO.	59, 114
$S_{\omega,max}$	Maximum value use for S_{ω} in PSO.	114
$S_{\omega,min}$	Minimum value use for S_{ω} in PSO.	114
$\mathbf{S}_x$	Position for one particle in PSO.	43, 45, 47, 48, 88, 101, 106
S_{Xmax}	Maximum position in the initialization of the PSO algorithm.	45, 107, 114, 123
S_{Xmin}	Minimum position in the initialization of the PSO algorithm.	45, 107, 114, 123
t	Time representation in iterative algorithms.	26
T_C	Computation time for a task/job to be scheduled.	16, 17, 21, 22, 118, 129
T_D	Absolute deadline for a task/job to be scheduled.	107
T_d	Relative deadline for a task/job to be schedule from the .	16, 22, 99, 118, 129
T_i	Task/job to be scheduled.	15, 22
T_L	Laxity or slack available for a task/job to be scheduled.	16, 22
T_N	Number of tasks to be scheduled.	123, 140
T_p	Period for a task/job to be scheduled.	26
$T_{penalty}$	Penalty instilled on the schedule for missing a deadline.	105, 106

t_{p_index}	Time within the defined schedule for the p_index^{th} processor.	105, 107
T_S	Start time for a task/job to be scheduled.	16, 21, 22, 99, 105, 107, 110, 118, 129, 130
T_{SA}	Temperature in degrees Kelvin used in SA.	35, 58
T_{set}	Set of tasks to be scheduled. $T_{set} = \{T_1, T_2, \ldots, T_N\}$.	106
$T_{tardiness}$	Tardiness for a given task.	105, 106
$\mathbf{x}$	Vector to be optimized.	32

Chapter 1

Introduction

The Engineering method can be defined as "the strategy for causing the best change in a poorly understood or uncertain situation within the available resources"

– Billy Vaughn Koen [Koen85].

Engineering problems often require optimal solutions that aim to maximize the outcomes, while minimizing the use of available resources. In some applications, the problems can be simplified into well defined models with distinct properties that can be solved analytically; however, most nonlinear problems require heuristics in order to find suitable solutions. This thesis presents a study of a family of *particle swarm optimization* (PSO) algorithms designed to address problems in emergent and incompletely defined environments [Forr91] like the performance of satellites in orbit [HaLo05]. The objective is to characterize the behaviour of the algorithm to improve the search for solutions in symmetric multiprocessing scheduling for space applications.

1.1 Problem Statement

There are many approaches to solve an optimization problem depending on the context to which it is applied. This section provides an overview of the classification of the environments for the problems, the modelled objective functions, and the specifics of the spacecraft systems addressed in this thesis.

1.1.1 Motivation

In 2009, *Magellan Aerospace Winnipeg* (MAW) commenced work on a generic satellite bus *command and data handling* (CDH) unit on a *printed-circuit board* (PCB), featuring up to four symmetric soft-core multiprocessors in a *field programmable gate array* (FPGA) [Nell11]. The reconfigurability of the FPGA increases the flexibility of the design and makes it possible to reuse for multiple missions through soft reconfigurations. This design reduces the cost and improves the operational response time for future satellites that utilized the generic satellite bus, *MAC-200* [PaHa06]. The additional computational power can be used for many tasks including (i) automating the *attitude determination and control* (ADC) functions of the spacecraft in order to reduce the number of reference parameters uploaded daily for orienting the spacecraft, (ii) compressing payload data to optimize the use of the throughput budget, and for (iii) pre-processing sensor and payload data. Thus, the main benefit of such operation would be to get a greater return-on-investment from the payloads. The hardware improvements are complemented through the use of customized real-time software that is able to respond to external and internal events faster than the time constraints established for the system [ScKM11]. Since there is no tolerance for missed deadlines, the system is considered to operate in hard real-time, while other applications can perform in soft real-time and thus accept some delays in the execution. In the spacecraft, the parallel software is executed in a *real-time operating system* (RTOS) [Chen02], capable

of working in any of the unit's configurations by scheduling tasks that maximize the use of available resources.

Typically, RTOS scheduling algorithms use either predetermined sequences of tasks or strict rules for predicable behaviours that aim to reduce the number of missed deadlines in the system. Since the new MAW reconfigurable platform can adapt to different system configurations when components or subsystems degrade over time, an *evolutionary algorithms* (EA)-based scheduler can be used to adjust the schedule to match the hardware needs. This adaptation to emerging system configurations is needed in uncertain environments like *low-earth-orbit* (LEO), where space debris, radiation, variability of the Earth magnetic field, and other unexpected conditions can affect a spacecraft's subsystems, causing it to operate differently than anticipated. As a result, the CDH unit, responsible for controlling the operations of the satellite, must adapt to the new states in order to keep the mission alive.

The EA-based scheduler requires a cost function to evaluate and compare the fitness of candidate solutions over time. The cost function design exploits behaviours of the evolutionary processes in order to minimize the computations, improve the rate of convergence, and therefore reduce the number of iterations required to find a solution. This is possible through novel characterization methods developed in this thesis that help visualize the effect of various parameters and their impact on the performance of the algorithm. These results aim to demonstrate the adaptability of the algorithm to different situations and demonstrate the feasibility of a RTOS scheduling algorithm based on EAs.

1.1.2 Problem Definition

Scientific and technology demonstration satellite missions aim to perform experiments in orbit, collect data, and transmit the data to the ground for analysis. However, for many

missions, the design of the satellite bus that carries and supports the science payload(s) can become a primary focus for development, delaying the launch, and increasing the cost of the overall mission [PrZe09]. The responsiveness (how long it takes to produce a satellite from the time the need is identified) is even more critical for commercial and military applications where the services provided by the payload(s) have a direct impact in the day-to-day lives of many people. Therefore, in an effort to accelerate the spacecraft development process organizations began investing in the design of generic satellite bus for use in multiple missions [SpYo89], [DePW04], [PrZe09].

In an effort to accelerate the process, organizations like the *Canadian Space Agency* (CSA) began working on the development of a generic small satellite bus for use in multiple missions over a period of ten years [PeBr04], [DePW04]. Small-satellites refers to the class of spacecrafts that are 500 kg or less [SSNK09]. In Canada, MAW was selected as the primary contractor and developed the MAC-200 satellite bus for use in future LEO Canadian missions [DePW04], [PaHa06], [HaKo10].

Magellan's MAC-200 features an expandable CDH composed of different cards used to control different components in the satellite [DePW04], [PaHa06]. The controller card on the CDH hosts a PowerPC processor running the VxWorks RTOS where the *flight software* (FSW) is executed [DePW04]. The FSW exhibits a multi-layered architecture that separates lower-level hardware interfaces and protocols from high-level tasks running on the RTOS. Since its inception, the MAC-200 has been used for the *CAscade, SmallSat and IOnospheric Polar Explorer* (CASSIOPE) mission [PaHa06], and is currently being employed for the *Outer Radiation Belt Injection Transport and Loss Satellite* (ORBITALS) [WeBe06], *Radarsat Constellation Mission* (RCM) [BeCa06], and Chinook missions [HaMB06]. However, as the end of the ten year expected life-span of the satellite bus approaches, MAW is exploring ways to improve the MAC-200 in order to continue to provide a high performance, scalable, reliable, and cost effective computing system for future satellite missions.

For the next generation of the MAC-200, MAW is switching to a FPGA that can be reconfigured dynamically to accommodate the requirements for future missions and reduce the development cost and verification time required for new printed circuit boards. To expand the processing capabilities, the FPGA is designed to host up to four LEON3 processors running a single shared instance of the *symmetric multiprocessing* (SMP) version of the open-source operating system RTEMS [Nell11]. The extra processing capabilities enables future spacecrafts to perform more on-board computations such as automated ADC algorithms, autonomous controls, and data pre-processing to optimize the use of downlink channels.

The added flexibility in the CDH poses many challenges for the scheduling algorithm in the RTOS [Dvor09]. During the design phase, the scheduling can be customized for a particular application; however, once in orbit, a satellite must be able to adapt to the changes in the environment, such as faults in a subsystem or processor failures. Furthermore, given two or more processors, there is no known deterministic algorithm for scheduling a set of independent and aperiodic tasks. Therefore, evolutionary optimization algorithms are proposed as a means of scheduling tasks in a generic satellite bus while also adapting to new on-orbit conditions that were not anticipated during the design phase.

1.1.3 Proposed Solution

The proposed solution is to characterize the trajectories of particles in PSO for scheduling tasks in a RTOS on a SMP architecture. The novel extraction and characterization of the trajectories in the time and frequency domain help select parameters for the algorithm to help increase the speed of convergence [ScKi11a]. Furthermore, this approach helps to visualize the evolution of the solutions and the behaviour of the particles when using different parameters in PSO.

In order to characterize the behaviour of PSO, the trajectories of the particles are extracted and examined. Since there are many different particles traveling along multiple dimensions, and the patterns are different for each run of the algorithm, an ensemble of particle trajectories is generated to provide statistical significance for the characterization. The trajectories are tested for stationarity and then long-term behaviours are investigated in both the time domain using correlations and in the frequency domain using power spectrums. These tests expand the qualitative work by Kennedy that identified two different types of behaviours in particles [Kenn97]. This technique helps to visualize the sensitivity of key parameters in the algorithm and their effects on finding the optimal solution. This characterization extends the known metrics to evaluate PSO.

A custom scheduling algorithm based on PSO is evaluated. The particle trajectories are extracted and analyzed to identify the effects of different parameters and how they can be used to improve the algorithm. The scheduler uses a mapping technique that separates the task allocation from the continuous trajectories of particles, thus not altering the principles behind PSO. Furthermore, the implementation provides a framework to add additional task properties such as precedences, processor affinities, and priorities to use the schedule for different applications. In addition, similar extensions can be created to add features like load shedding into the scheduler. This implementation is tested in our simulator that enables the results to be analyzed numerically and visually.

1.2 Thesis Formulation

1.2.1 Thesis Statement

This thesis aims to develop a scheduling algorithm for aperiodic and independent tasks in SMP for hard real-time applications, like small satellites, using PSO through an analysis

of particle trajectories in both the time and frequency domain that aid in the design of the scheduler.

1.2.2 Thesis Objectives

The thesis has two main objectives:

1. Identify implementation independent measures for the performance of EAs, including:
   (a) Implement the PSO algorithm;
   (b) Extract implementation independent measures for PSO; and
   (c) Experiment with different classes of functions to establish the performance of the metrics for different problems.
2. Design a scheduling algorithm based on PSO for real-time SMP running aperiodic and independent tasks, including:
   (a) Optimize the algorithm to reduce the total length of the schedule produced while meeting all the hard real-time requirements;
   (b) Provide a framework for extending the algorithm to incorporate other features like load shedding (not included in this study because it cannot be verified/tested in the simulator); and
   (c) Test the algorithm performance using numbers of tasks and system loads that are representative of existing satellite missions from MAW.

1.2.3 Research Questions

The thesis presents a number of research questions on evolutionary algorithms and their applications to real-time systems. The following list of questions highlights some of

the major topics and describes which ones are addressed in this thesis.

1. What characterizes the performance of EAs?

   (a) What measures should be used to characterize the behaviour of a swarm?

   (b) What are the topologies for inter-particle communication in a swarm?

   Comment: EAs rely on complex interactions that make up the evolutionary processes. Understanding these interactions and their impact can help improve the performance of these algorithms. This thesis attempts to address the question for PSO by observing the trajectory of individual particles in a swarm to observe the impact from different topologies. This enables further research to identify the significance of the extracted trajectories in order to establish methods to optimize parameters in the algorithm for a particular application.

2. How can one characterize the quality of behaviour in a swarm?

   (a) What are the differences in the behaviour of continuous versus discrete swarms?

   Comment: PSO guides particles towards a solution. The movement is intuitively characterized by a convergence of all the particles around a particular area surrounding the solution. To expedite the algorithm, one needs to understand the macrocosms of behaviours exhibited by the swarm in order to distinguish good behaviours in the swarm that can indicate the quality of the solution and whether the parameters need to be modified at runtime in order to obtain a solution. These questions are not addressed in this thesis, however insights are provided based on the observed behaviours at the microcosm level of individual particle trajectories.

3. What are the implementation independent measures to evaluate EAs?

(a) What are the metrics for PSO?

(b) Can we measure/capture the behaviour of the entire swarm in order to compare different implementations?

(c) What is the optimal granularity of partitions to use when looking at the convergence of an ensemble of particles over time?

Comment: Typically, EA implementations are compared based on the number of iterations to solve textbook examples up to a predetermined precision. The iteration count depends on the specific implementation and fails to account for the complexity of the algorithm, length of each iteration, and evolutionary processes that aid in the search of solutions in real applications. Therefore, it is important to investigate metrics that are implementation independent and provide an insight into the behaviour of the algorithm, thus serving as a method for comparison for different implementations and the selection of the associated parameters. This thesis addresses the problem through a novel trajectory analysis that show the emergent behaviour, exploration of the parameter space, and convergence towards a solution.

4. What is the sensitivity of the parameters in PSO?

(a) What is the relative importance of the parameters?

(b) What is the range of values each parameter can take to guarantee convergence?

(c) What are the interdependencies between parameters?

(d) What type of erratic behaviour from a particle can affect the convergence?

(e) How many particles need to behave erratically in order to affect convergence?

Comment: The PSO algorithm uses many parameters that affect the movement of

particles towards the solution. Many researchers have studied the parameters both analytically and experimentally to establish constraints and heuristics to help in the selection process based on convergence analysis. However, there exists very little information on the coupling effects of the parameters, the effects on the behaviour of individual particles, and the overall effect on the behaviour of the swarm. This thesis provides a framework for analyzing the interdependencies of the parameters in PSO through the particle trajectories. Insights into the effect of the different parameters is provided, however in depth research is required to fully comprehend the sensitivity of the parameters.

5. Can evolutionary optimization algorithms be used for scheduling applications in multi-core real-time systems?

   (a) What is the scalability of the algorithm?

   (b) How could the algorithm handle sporadic tasks, aperiodic tasks, or interrupts?

   Comment: The core motivation of the thesis is to apply evolutionary algorithms to schedule hard real-time applications in multi-core real-time systems in order to generate schedules in emergent environments like satellite in LEO. This thesis addresses the question through the design, implementation, and testing of a PSO-based scheduling algorithm. In particular, the thesis focuses on independent and aperiodic tasks. Sporadic tasks with low frequency of occurrence and interrupts are not addressed in this study.

1.3 Thesis Organization

This thesis presents a scheduling algorithm for SMP real-time systems using PSO. The performance of the scheduler is evaluated by studying the trajectories of individual particles

as they move through the scheduler cost function looking for a solution. Chapter 2 provides an overview of scheduling algorithms and evolutionary algorithms aimed at identifying key characteristics and terminology used throughout the thesis to compare and analyze the work done. Chapter 3 provides an in-depth review of the original PSO algorithm and the significant improvements that aid in the understanding of the trajectories. Chapter 4 highlights a methodology for extracting trajectories from the particle optimization algorithm, verifies the PSO performance against benchmark examples, and describes the cost function used for scheduling real-time tasks in SMP. Chapter 5 focuses on the implementation and testing environment of the scheduling algorithm based on PSO. The results of the work are described in Ch. 6 and main conclusions are stated in Ch. 7.

Chapter 2

Literature Review of Scheduling Algorithms

Scheduling in real-time systems is the process of deciding when to execute each task on the system and, in the case of multiprocessor systems, which processor to assign a task to. Unfortunately, most problems are classified as *NP-complete* (NPC), with only a few special cases with polynomial-bounded scheduling algorithms available for dealing with aperiodic and independent tasks . One approach to address large class of problems is to use evolutionary algorithms that produce a set of possible schedules and iteratively improve them to find a solution. This chapter discusses the main classes of scheduling algorithms and evolutionary algorithms that serve as the foundation for comparison in this thesis.

2.1 Scheduling Real Time Systems

Real-time systems must respond to external and internal events faster than the time constraints imposed on the system [Kins12]. A distinction can be made for *hard* real-time

where there is no tolerance for missed deadlines, and *soft* real-time where some delays are acceptable. Thus, real-time systems are evaluated not only on the result of a computation, but also the time at which the result is provided [RaSt94]. The timeliness of results is addressed through proper *scheduling algorithms* that decide the order in which to execute tasks and, in the case of multiprocessor systems, which processor to assign the task to.

The study of scheduling algorithms covers a wide range of fields, and the definition of the tasks to be scheduled, procedures, and constraints are very different for each application. For example, in *operations research*, scheduling often refers to arranging tasks in a manufacturing process such as assembly lines [CoMM67]. In this context, each product being assemble in a flow-shop is identified as a task, and has a number of operations that must be performed in a particular sequence, on specific machines, in order to obtain a finished product. The objective of such operations is to improve the throughput to maximize the profit from the operation and is often illustrated in the form of *PERT charts* showing the interdependencies between tasks that help identify the critical path [WeLa99].

In contrast, in *logistical operations* such as military activities, scheduling refers to the supply chain management of crisis situations including the transportation means and distribution of resources [KZHZ11]. The objective of such operations is to deliver the necessary supplies to the target areas as quickly as possible to minimize civilian casualties, with the severity of delays varying for each problem. In these cases, synchronization, knowledge sharing, and cooperation are the primary objectives.

Although there are many similarities in scheduling across different fields, the process of scheduling real-time systems is unique in the severe consequences of missed hard deadlines. The following sections expand the scheduling problem for real-time systems.

2.1.1 Scheduling Complexity and Optimal Schedulers

Intuitively, scheduling is an increasingly difficult problem as more tasks and constraints are introduced. The general problem is classified as NPC [RaSt94]. In general, given sufficient time and resources, these problems can be solved by polynomial-depth backtrack search algorithms that recursively test different solutions and cut branches where constraints would always be violated [Chen02]. Since the problem is very common and has many day-to-day applications, there is a need to find heuristics and develop algorithms that can produce good solutions, if not optimal, in relatively short periods of time. This can be accomplished by narrowing down the scope of the general scheduling problem by defining parameters and constraints (as those outlined in Sec. 2.1.2) in order to find algorithms for specific types of applications. For example, the *largest local remaining execution time first* (LLREF) scheduling algorithm is designed to work with periodic tasks whose deadline is the beginning of the next period has an execution time of $\mathcal{O}(N^2)$, while the extension to the *local remaining execution time and local execution time* (LRETL) improves the run time to $\mathcal{O}(\log T_N)$ [ChRJ06] [FuNa10]. Furthermore, the book by Richard Conway, William Maxwell, and Louis Miller contains many tables identifying key parameters and constraints for specific instances of the scheduling problem with references to the algorithms developed to solve those problems in polynomial time [CoMM67], with more recent works described by Peter Brucker in [Bruc95].

In this context of application specific schedulers, one can define an *optimal scheduler* as one that *"may fail to meet a deadline of a task only if no other scheduler can"* [CoMM67], [Chen02]. From this definition, there can be more than one optimal scheduling algorithm for a particular problem, and therefore the *asymptotic algorithm complexity* (AAC) (sometimes referred to as *big-O notation*) can serve to identify the better candidate for a given problem. As such, the tables listed in [Bruc95] provide the computational complexity of each of the

algorithms described to provide a reference point to users on both the implementation and performance of the algorithm.

The AAC estimates are based on the set of rules used to generate the schedule and often include searching for tasks with key properties to be executed next. These types of algorithms work well in deterministic environments and are fully reproducible. However, the AAC estimates become more difficult to estimate when incorporating evolutionary algorithms because of the stochastic elements in the algorithms. Therefore, many authors do not provide algorithmic complexity measures for evolutionary schedules and, instead, uses other metrics, such as the number of iterations to convergence, to establish the relative merits for the different algorithm implementations.

2.1.2 Scheduling Problem Definitions & Classifications

The generic scheduling problem is often described in terms of four parameters: (i) the tasks to be processed, T_i $(i = 1, \ldots, T_N)$, (ii) the number and type of machines executing the tasks, C_j $(j = 1, \ldots, C_N)$, (iii) the restrictions on task assignments and (iv) the criteria used to evaluate the schedule [CoMM67], [LeRB77].

2.1.2.1 Tasks

Tasks are classified based on their behaviour. *Single-instance* tasks execute only once and are common of initialization routines or termination routines that setup hardware for particular applications [Chen02]. *Periodic* tasks have many instances separated by a fixed time between successive instances, while *aperiodic* tasks do not have a fixed separation time [Chen02]. Periodic tasks are commonly used for sampling information about the environment such as to provide temperature profiles. Finally, *sporadic* tasks have zero or more occurrences with a minimum separation between consecutive releases [Chen02].

These are regularly used for handling anomalies in systems such as emergency maneuvers in aircraft.

The task is described in terms of three parameters $T_i = \{T_S, T_C, T_d\}$ denoting the *start-time* or *arrival-time*, T_S, the *(maximum) number of computational units*, T_C, and a *relative deadline from the start-time*, T_d. Figure 2.1 shows a sample task, T_1 to be scheduled. A rectangle along the horizontal axis shows the execution parameters of a task, including the start time, T_S, the worst-case computation time, T_C, and its relative deadline, T_d. The rectangle width goes from $t = T_S$ to $t = T_S + T_C$ representing the start time and computation time respectively. A downward pointing arrow is placed at $t = T_S + T_d$ to denote the corresponding relative deadline [DeMo89]. The gap between the task completion time and the relative deadline is known as the *laxity* or *slack*, T_L, and is defined as $T_L = T_d - T_C$ [Chen02].

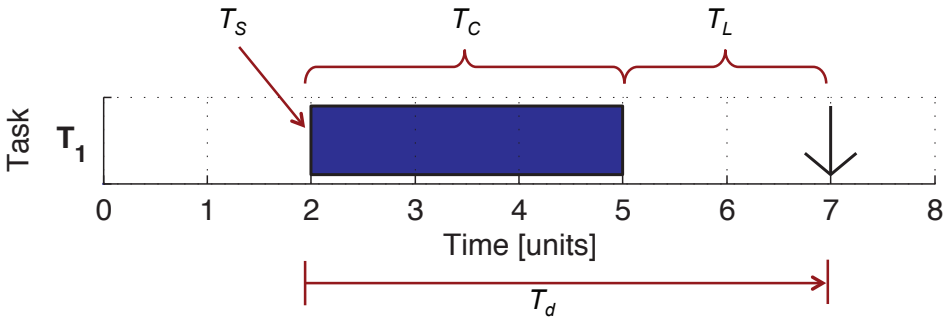


Fig. 2.1: Timing diagram showing task $T_1 = \{T_S, T_C, T_d\} = \{2, 3, 5\}$.

Furthermore, in some applications extra parameters are added to tasks such as *priority*, *period*, and *predecessors* [LeRB77]. The priority is used to bias the scheduling process to ensure critical tasks with hard deadlines are processed first before addressing some background processes. The period is exploited in scheduling periodic tasks by reserving resources for certain tasks by shifting the surrounding aperiodic tasks accordingly. Finally, the predecessors help dictate the order in which tasks must execute for a particular application.

2.1.2.2 Computer Architecture Configurations

The computer architecture has significant effects on the design of scheduling algorithms. The number and type of processors plays a major role in the design, while other factors are derived from the memory configurations, shared resources, and peripherals.

The single processor (or uniprocessor) architecture consists of one processor, memory, and the necessary peripherals. In this configuration, all processing and data handling passes through the same microprocessor. The memory access time has the potential to delay some key operations, therefore worst case estimates are often included in the computational unit measures, T_C, for each task [Liu00] [Chen02] [CoFe13]. If *direct memory access* (DMA) is used, the access time can often be reduced, but special attention must be paid to preemptive schedules (as defined in Sec. 2.1.2.3) to ensure the integrity of the data is maintained. This is often done through semaphores to protect key pieces of code containing large data transfers that are accessed by multiple tasks on the system. Peripherals are configured as either *memory mapped* or *input/output* (IO)*-mapped* [Kins12]. The memory mapped systems share the bus architecture with the processor and use reserved addresses to access the internal registers in the devices, while IO-mapped devices use dedicated processor lines (either parallel or serial) for communication. The schedule must pay special attention to peripherals that generate interrupts in order to accommodate the functions within the real-time constraints. Models of the worst-case occurrences of interrupts are incorporated into the schedule to ensure that if the processor is servicing external devices it can still complete the regular tasks to maintain the real-time requirements of the system [Chen02].

In multiprocessor architectures, tasks not only need to be scheduled, but also assigned to a particular processor. Thus, the increased number of processors, C_j, provides more choices where tasks can be assigned and therefore, increase the possible schedules produced. The processors can be homogeneous to produce a SMP or heterogeneous for *asymmetric multi-*

processing (AMP). In SMP, there is no advantage as to where each task is assigned, making it a redundant architecture. In contrast, AMP focuses on exploiting specific features of each processor to speed up the overall system or perform specialized tasks utilizing features of different processors. In both configurations, the memory and other shared resources can be either connected to an individual core or the whole system. Shared memory architectures must deal with special cases to ensure the same addresses are not used by more than one processor at a time, while independent memories pose other challenges by introducing processor affinity requirements (as defined in Sec. 2.1.2.3) onto the system. Similarly, external devices can be connected to individual cores or to the whole system in order to handle IO connections as well as incoming interrupt calls. If connected to individual cores, the interrupts are managed within the associated processor, however in other cases, additional logic is required to determine which processor can handle the interrupt without causing deadlines to be missed.

2.1.2.3 Scheduling Constraints and Heuristics

There are many constraints imposed on scheduling algorithms depending on the particular application. Furthermore, there are many heuristics used to optimize the performance of the system given *a priori* knowledge of the tasks and architecture used. The following list defines some of the major constraints and heuristics with their associated benefits for particular scheduling applications.

Static, Hybrid, and Dynamic Schedules

Static schedules are computed *a priori* and are fixed for a particular application. The scheduler runs through a sequence of tasks in a predefined order that is designed to satisfy the real-time requirements of the system. Interrupts are commonly implemented as short functions that setup flags for the schedule to handle request

when the appropriate function is executed. In contrast, *dynamic schedules* are calculated at runtime based on the properties of the tasks available. This process requires more processing resources allocated to the scheduler in order to determine the operations to perform. The *hybrid schedules* are calculated whenever the system changes state, but are considered static otherwise. This combination retains the flexibility of dynamic schedules while reducing the computational resources required. Examples of different schedules are provided in Sec. 2.1.5.

Processor Affinity

In multiprocessor architectures tasks can have an *affinity* with a particular processor to either interface with its associated external devices or take advantage of special features of a processor in AMP. This is easy to achieve in static or hybrid schedules as the constraints can be pre-computed to ensure the system requirements are met. However, when working with dynamic schedulers, the task's properties need to be considered to help make probabilistic decisions on whether or not to reserve the processor in question or whether the task has enough slack to run another operation on that processor.

Preemption

Preemption or *time-shared* refers to the process of stopping the currently running task, switching the context of the machine, and finally, starting another task with higher priority or a more immediate deadline. Dynamic schedules with preemption can interrupt current tasks based on new arrivals to achieve the best results, while those without preemption can only change the schedule upon completing the current operation. The cost of preemption is the time and memory required for context switching, and thus some applications may limit the levels of preemptive tasks to reduce overhead operations. Finally, in multiprocessor systems, preemption algorithms must also assess which of the processors to interrupt for a particular

application.

Migration

In a multiprocessor system with preemption, tasks without affinities to a specific processor can start in one core, pause, and *migrate* to another core to continue executing. Migration is difficult to achieve in AMP because the internal state of the task saved in memory may not be able to run on a different processor. However, task migration can exploit the redundancy in SMP that can move tasks off a malfunctioning processor, or to a processor which has more free resources.

Task Precedence

In applications where the task operations follow a predefined sequence (i.e., data processing), the scheduler must take into account the order of operations to ensure that *task precedence* requirements are properly met. Multiprocessor systems must use special messages between processors to ensure that the sequence of tasks is maintained when distributing parallel operations through multiple cores.

Task Priority

Scheduling algorithms must take into account the *priority* of tasks to ensure that all critical operations are completed first. The priorities can be either static or dynamic depending on the application. In multiprocessor systems, priority handling takes into account the number of processors in order to select, and if needed preempt, tasks such that the global priorities are maintained.

2.1.2.4 Criteria for Evaluating Schedules

The primary criterion for real-time systems is that all tasks meet the deadlines. For that to be possible, the sum of utilization of all tasks must be less than or equal to the processor resources. Furthermore, a schedule is valid if the following conditions are satisfied

[Liu00]:

1. Every processor is assigned to work on at most one task at a time.
2. Every instance of a task is assigned to at most one processor at a time, except for redundancy voting architectures where tasks are computed more than once to compare the results and overcome both hardware and software failures.
3. No task is scheduled before its start or release time, T_S.
4. The total amount of processor time allocated to every task must be at least equal to the computation time, T_C, for the task.
5. Task precedence, processor affinity, and priority constraints are satisfied.

Even with all these conditions, a scheduling algorithm may produce two or more different schedules for the same application with the same set of tasks. In static schedules it is sufficient for a schedule to meet all deadlines and constraints for the system. In contrast, in dynamic systems with some *a priori* knowledge of the tasks, there can be multiple schedules that meet the deadlines. In such situations, secondary metrics may be employed to determine which schedule to use. Examples include: (i) minimize schedule-length, and (ii) minimize average tardiness. In these terms, *tardy* refers to small delays that use the slack time for a task and still meet the deadline, while *late* indicates the task has missed its deadline. In both tardy and late tasks, the *effective start time* refers to the actual time when the task begins executing. Note that techniques such as measuring *fairness* of the resource distribution to prevent *task starvation* is not suitable for hard real-time systems because this information is embedded in the requirement that every task meets its deadline.

When generating schedules with evolutionary algorithms, the criteria is particularly important as it serves as the means for optimization of the algorithm. As solutions are generated, they are evaluated based on the criteria defined with additional parameters to

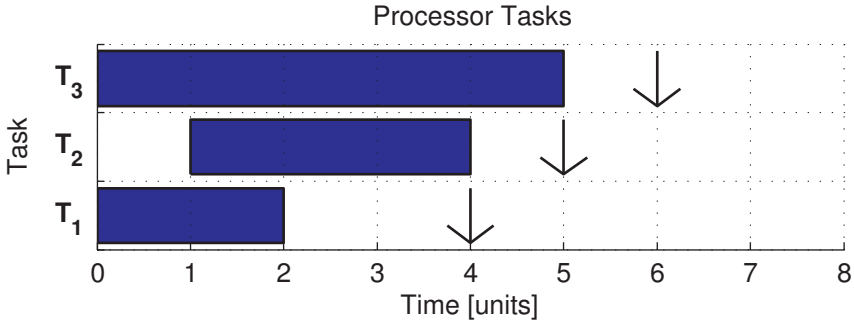
account for deadlines met or missed.

2.1.3 Visualizing Schedules

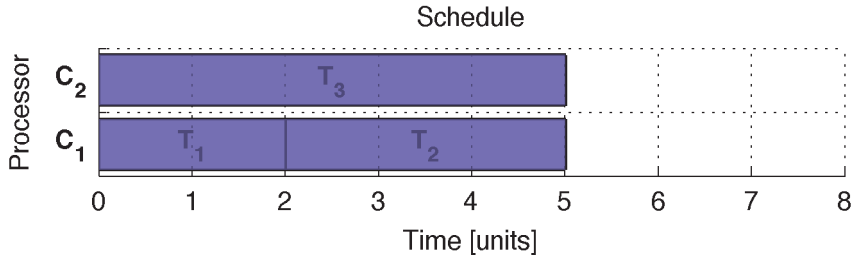
In developing scheduling algorithms, it is helpful to visualize the schedule in order to gain an intuitive understanding of the algorithm performance. The most common representation for schedules is in the form of a *timing diagram* that shows time elapsed along the x-axis and represents each processor on a separate row as shown in Fig. 2.2(b). Each task, T_i, is represented by a rectangle that begins at the effective start time for the given task and lasts T_C time units on the corresponding processor row, C_j. Each row contains the tasks executed by the processor in the order in which they are serviced for a given schedule. If preemption is allowed, the rectangles may be broken down into smaller pieces whose width would total an execution time of T_C time units.

This timing diagram can provide an instantaneous representation of the schedule which is easy to describe, but it fails to provide information about the real-time nature of the system. More specifically, it does not show the start time, T_S and the relative deadline, T_d. The basic timing diagram shows a schedule but does not indicate whether it is valid or not. To show validity, colours can be added to highlight tasks with missed deadlines as shown through the red tasks in Fig. 2.2(c).

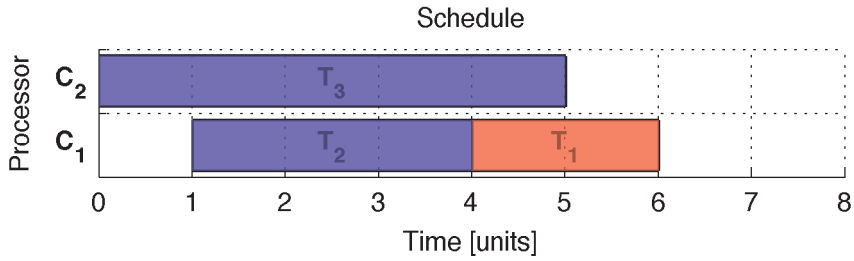
Dertouzos and Mok developed an alternative representation of schedules known as the *scheduling game board* to show a dynamic representation of the status of the tasks at each instance in time [DeMo89]. The model is specifically suited for SMP and can verify that all tasks complete their execution by their respective deadline. The scheduling game board shows the laxity, T_L, plotted on the x-axis and the *remaining* computation time, T_C, on the y-axis. Each task is represented by a circular token on the T_L-T_C plane. At each time step, each of the C_j processors executes one task and moves it downward one unit corresponding to



(a) Timing diagram of three tasks to be scheduled.



(b) Valid schedule for tasks that meets all real-time deadlines.



(c) Invalid schedule where task T_1 misses its deadline.

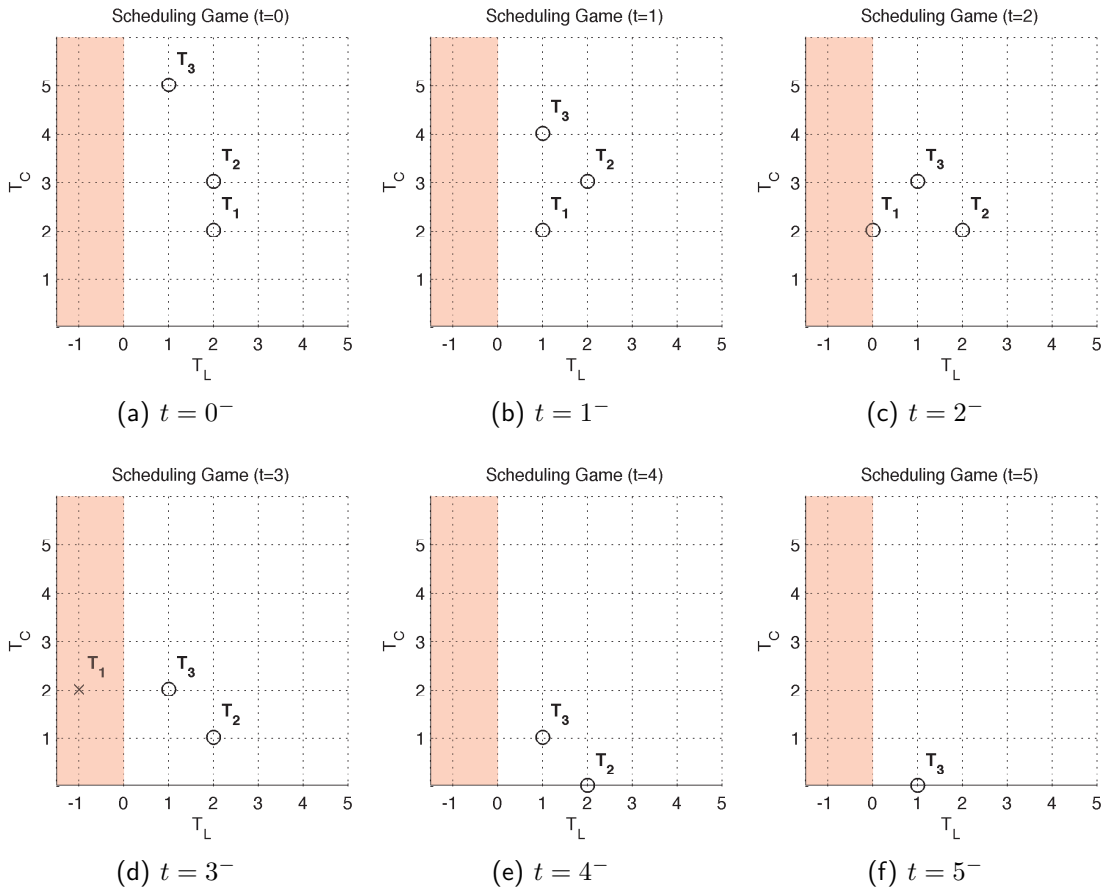
Fig. 2.2: Sample timing diagrams with possible schedules selected to show the differences between valid and invalid schedules. (a) The three tasks to schedule defined as $T_1 = \{T_S, T_C, T_d\} = \{0, 2, 4\}$, $T_2 = \{1, 3, 4\}$, and $T_3 = \{0, 5, 6\}$. (b) A valid schedule for $C_N = 2$ processors. In this schedule, T_1 terminates at $t = 2 \leq T_{1,d} = 4$, T_2 ends at $t = 5 \leq T_{2,d} = 5$, and T_3 completes at $t = 5 \leq T_{3,d} = 6$. Thus, all three tasks meet their corresponding deadlines. (c) An invalid schedule where task T_1 missed its deadline because in this schedule it is executed after T_2.

a reduction in the remaining computational time. The remaining tasks are moved leftward one unit as their laxity decreases making the execution of the task more urgent. Tasks reaching the horizontal axis can be removed from the game as they have completed their execution. However, any tasks crossing the vertical axis before reaching the horizontal axis fail to meet their deadline. For a given configuration, one can select which tasks to execute based on a scheduling algorithm and therefore influence the performance of the system. In this context, the game is won if no deadlines are missed. Figure 2.3 shows the same set of tasks from Fig. 2.2(a) depicted with the scheduling board game the instance before each time step begins (i.e., $t = 0^-$ is initial representation before the tasks are executed at $t = 0$). The tasks are executed according to the invalid schedule from Fig. 2.2(c) to demonstrate inactive tasks, missed deadlines, and tasks completed.

2.1.4 Schedulability Analysis

As highlighted by the scheduling board game, it is important to analyze whether there is a solution to the problem that will meet the deadlines for a particular system. In some cases this can lead to changes in the constraints for the scheduling algorithm, the task implementation, and even the system as a whole. This analysis, or *schedulability test*, serves as a necessary and sufficient condition for a feasible schedule given a set of tasks, the number of processors, and some of the constraints [Chen02]. In general, a schedulability test attempts to define an inequality to evaluate whether the available processing resources are sufficient for the given set of tasks. If the inequality is satisfied, there exists a feasible schedule, but it is up to a scheduling algorithm to identify that solution.

There are many examples of schedulability tests depending on the constraints for the system and tasks properties. For uniprocessors, tasks are often modelled as both preemptable and independent (i.e., it does not have any precedence relations or constraints from



(a) $t = 0^-$ (b) $t = 1^-$ (c) $t = 2^-$

(d) $t = 3^-$ (e) $t = 4^-$ (f) $t = 5^-$

Fig. 2.3: Scheduling game board using the tasks from Fig. 2.2(a) and the same failed schedule from Fig. 2.2(c) for the case of $C_N = 2$. (a) Initial placement of the tasks at $t = 0^-$. (b) Task T_1 is ready but does not execute, so its laxity decreases by 1 unit, task T_2 does not move because its start time is greater than $t = 1^-$, and task T_3 completes 1 unit of computations. (c) Tasks move based on game rules. T_1 reaches a critical point where if it is not executed it will miss its deadline. (d) T_1 misses its deadline. (e) T_2 completes its computation. (f) T_3 completes its computation.

shared resources) in order to estimate whether the available processing resources can complete all the tasks before their deadline. Details on these uniprocessor tests can be found in [Chen02] and [Liu00]. In multiprocessors, there are two major tests to evaluate conflict-free tasks using both single instance and periodic tasks. These are described next.

This tests divide the scheduling board game into three regions defined by Eq. 2.1 to

separate tasks that require immediate attention ($R_1(t)$), tasks that must be serviced within the next t units ($R_2(t)$), and tasks not considered immediate ($R_3(t)$).

$$R_1(t) = \{T_i : T_{i,D} \leq t\} \tag{2.1a}$$

$$R_2(t) = \{T_i : T_{i,L} \leq t \wedge T_{i,D} > t\} \tag{2.1b}$$

$$R_3(t) = \{T_i : T_{i,L} > t\} \tag{2.1c}$$

Given these definitions, a *surplus computing power function*, $R_A(t)$, is defined for t units into the future, as shown in Eq. 2.2. The first term of this equation estimates the total computational units available assuming a SMP architecture over t time units. The estimates for the computational units from $R_1(t)$ and $R_2(t)$ tasks are subtracted to identify the surplus power available. Note that the $R_3(t)$ tasks are not included as they do not need to be executed within the future t units for the system to meet its hard real-time deadlines.

$$R_A(t) = tC_N - \sum_{T_i \in R_1(t)} T_{i,C} - \sum_{T_i \in R_2(t)} (t - T_{i,L}) \tag{2.2}$$

Then, the necessary condition for scheduling real-time tasks is that for all $t > 0$, $R_A(t) \geq 0$, thus guaranteeing that the existing resources are sufficient.

The second test deals with scheduling periodic, preemptable, and independent tasks in a multiprocessor system. It first computes the total utilization of the processors and ensures that there are no conflicts with the periods, T_p, that would make the system run out of resources when executing a particular task in the future [Chen02]. This is demonstrated by first ensuring that all the tasks can be computed within the available resources as shown in Eq. 2.3.

$$R_{P1} = \sum_{i=1}^{T_N} \frac{T_{i,C}}{T_{i,p}} \leq C_N \tag{2.3}$$

Then, estimates whether the period of all the tasks can be scheduled without conflicts by ensuring that R_{P3} is an integer to guarantee that even if there are enough resources, the periods do not conflict as evaluated using the *greatest common divisor* (GCD) in Eq. 2.4. In essence, this equation attempts to schedule each task for $R_{P2}\frac{T_{i,C}}{T_{i,p}}$ time units (thus executing a full $T_{i,C}$ per task period).

$$R_{P2} = GCD(T_{1,p}, T_{2,p}, \ldots, T_{N,p}) \tag{2.4a}$$

$$R_{P3} = GCD\left(R_{P2}, R_{P2}\left(\frac{T_{1,C}}{T_{1,p}}\right), R_{P2}\left(\frac{T_{2,C}}{T_{2,p}}\right), \ldots, R_{P2}\left(\frac{T_{N,C}}{T_{N,p}}\right)\right) \tag{2.4b}$$

Both of these tests assume some *a priori* knowledge of the problem space that can be modelled with worst-case scenarios to assess the performance of the system and ensure a valid schedule is feasible. Even if a schedule is found to be feasible through these or other similar tests, the processor load should not run at 100% capacity as that fails to account for anomalies, some extreme behaviours, and system changes due to code maintenance. As such, some organizations employ heuristics for resource margins in processor load and how to manage interrupt handling in real-time systems [GOLD09].

2.1.5 Review of Scheduling Methodologies for Real-Time Systems

There are many approaches to develop algorithms for scheduling real-time systems including (i) clock-driven [Liu00], (ii) round-robin [Liu00], (iii) priority-driven [Liu00] [Chen02], (iv) queuing theory [Leho96], (v) Markov chain and hidden Markov chain [TBLW11], and (vi) evolutionary schedules [LiWJ07] [Cook10]. This section provides an overview of these types of schedules.

2.1.5.1 Clock-Driven Schedules

A *clock-driven* schedule makes decisions for executing tasks at predefined time intervals in a uniprocessor system [Liu00]. The scheduler is driven by an external periodic interrupt or external timer that triggers when the next task should start. If a task finishes its operations sooner than expected, the processor enters an idle mode until the next interrupt triggers the start of the execution for another task. The schedules are non-preemptive and static to minimize the overhead, thus making these schedules ideal for applications where precise sampling of the environment is required.

2.1.5.2 Round-Robin Schedules

Round-robin schedules are often used for time-shared applications to ensure fair and starvation-free dynamic schedules in uniprocessor systems without task precedence [Liu00]. In this setup, each task is entered into a *first-in-first-out* (FIFO) when it becomes ready for execution. At each time-slice (typically on the order of 10 $msec$), the top task in the queue is executed. If the task fails to complete its operations in the allocated time, it is placed at the end of the queue to wait for the next turn. This configuration cannot guarantee that all the real-time deadlines for the system are met, so a worst case analysis on the tasks is required to perform a schedulability test and determine the feasibility of the design. For multiprocessor systems, the schedule is augmented to a *weighted round-robin* that allocates time slices that are proportional to the task length [Liu00]. Furthermore, weighted round-robin implementations have also been used to separate tasks based on which processor should execute them in order to reduce the overall schedule length. This is accomplished by streamlining pipelined applications with tasks precedence and also manage task affinity requirements for both AMP and SMP [Liu00].

In both clock-driven and round-robin algorithms, a task can terminate before the end

of the time slice thus intentionally leaving the processor idle until the next time slot begins. In contrast, *priority-driven* or *event-driven* schedules make decisions when new tasks are released and when tasks finish executing. The decisions are based on priority schemes that maximize the use of available resources and, by definition, never leave a resource idle intentionally [Liu00]. This greedy methodology always executes the task with the highest priority and can therefore lead to suboptimal choices for the overall system performance (i.e., longer overall schedule). Thus, although the priority-driven schedules are very easy to implement and do not require much information in advance, they are not often used for safety-critical real-time systems because it is difficult to validate whether all the tasks can meet their deadlines [Liu00].

2.1.5.3 Priority-Driven Schedules

In priority-driven systems, the tasks can use fixed priorities, as in *rate-monotonic* (RM) and *deadline-monotonic* (DM), or dynamic, as in *early deadline first* (EDF) and *least-laxity first* (LLF) [Liu00]. In RM, the *rate* of releases (inverse of the period) is used as the priority for scheduling decisions [Chen02]. The shorter the period of a task, the higher the priority within the system. Similarly, the DM also deals with periodic tasks and assigns priorities based on the relative deadlines, so that shorter relative deadlines become more urgent within the system. These algorithms can be expanded to multiprocessor applications by conducting the analysis off-line to guarantee the tasks can complete the operations to produce valid schedules. In contrast, EDF and LLF are designed for non-periodic tasks and assign priorities based on the relative deadlines or laxity of the tasks respectively. The priorities are recalculated dynamically every time a new task is generated or completed [Chen02]. One major difference between these two algorithms is that the EDF does not require the computational time for each task, and thus it is easier to implement in many systems. There have been many attempts to expand EDF and LLF algorithms to multipro-

cessor systems, however the processor allocation heuristics remain an open research topic for the purpose of a generic algorithm.

2.1.5.4 Queuing Schedules

The scheduling methodologies discussed so far are designed to meet the real-time constraints and fail to evaluate the overall performance of the system. Therefore, rather than simply looking at the current state of the system and the top of the task queue, *queuing theory* examines the overall response time and throughput of the system with the objective of minimizing the mean waiting time for tasks and number of tasks waiting [Leho96]. The statistical analysis assigns probability distributions to the arrival time and execution time of the tasks instead of always assuming the worst-case scenarios like in priority-driven methods. Using this theory, [Leho96] expands the EDF algorithm to incorporate stochastic task properties for a uniprocessor system in order to show that the overall use of resources can be minimized for real-time systems at the expense of some additional overhead. Expansions of this model to multiprocessor systems are limited by the same problems encountered in priority-driven methods where the processor assignment adds an extra level of complexity that makes the problem intractable.

2.1.5.5 Markov Schedules

Similar to queuing theory, Markov processes attempt to look at the overall problem but make the decisions dynamically by estimating the reward or advantage associated with selecting each task [TBLW11]. This generates a graph showing the reward associated with each possible subsequent task. The graph is updated as tasks are completed, miss deadlines, or new tasks are generated. This is an extension to the priority-driven algorithm that can simulate multiple steps into the future to estimate what sequence of tasks offers the highest

reward. The reward helps to quantify the advantages gained by some scheduling policies given a set of tasks. The reward calculations and estimates into the future use different techniques such as (i) sequencing exhaustively for benchmarking, (ii) greedy heuristics that only consider the immediate effects like EDF, (iii) deadline heuristics that expand EDF for some iterations into the future to determine if the choices are indeed optimal [TBLW11]. Despite its benefits, Markov processes are computationally intensive and seldom used in hard real-time systems.

2.1.5.6 Evolutionary Schedules

Evolutionary schedules are iterative processes that start with a random solution, make perturbations to the solution, and have some mechanisms to accept or reject the changes in order to guide the search towards a solution. These algorithms are very suitable for large problem sets where testing every combination of schedules is not feasible. Each solution generated has a cost associated to it that represents how good a solution is with respect to another candidate solution. For scheduling, the cost function evaluates missed deadlines and often use additional secondary constraints to help guide the search. [LiWJ07], [Cook10]. Like queuing theory, this method takes time to evaluate as it must test various options in order to find a suitable schedule. For this reason, it is used for many soft real-time applications with limited utilization to find optimal schedules in static environments for real-time systems.

2.2 Evolutionary Optimization Algorithms

This thesis uses evolutionary algorithms to find solutions for scheduling problems. The search for the best solution is accomplished through optimization algorithms capable of de-

termining the course of action amongst many different alternatives available [Boyd04].

$$\begin{aligned} &\text{minimize} && f_{objective}(\mathbf{x}) \\ &\text{subject to} && f_{constraint}(\mathbf{x}) \leq 0 \end{aligned} \tag{2.5}$$

Optimization problems aim to find values for the vector of parameters $\mathbf{x} \in \mathbb{R}^D$ that minimize the objective function, $f_{objective}(\mathbf{x})$ subject to the constraints defined by $f_{constraints}(\mathbf{x})$ as shown in Eq. 2.5 [Boyd04]. The set of all possible values for $\mathbf{x}$ makes up the parameter space for the problem.

Solving the generalized problem described in Eq. 2.5 without an exhaustive search requires knowledge of the parameter space, and a cost function that provides a figure of merit between two candidate solutions [KeEb01]. These two pieces of information help select the appropriate approaches to accelerate an iterative search for optimal solutions by pruning portions of the parameter space. More specifically, using the *a priori* knowledge of the parameter space and cost function helps identify the class of problem and the key properties needed to select the appropriate technique to solve the problem.

Using the properties of the parameter space and the cost function, one can select the appropriate technique to solve a problem. For example, one can check if an objective function is linear [Boyd04]. Given vectors $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{R}^D$ and constants $\alpha_1, \alpha_2 \in \mathbb{R}$, a function is said to be linear if it satisfies Eq. 2.6.

$$f_{objective}(\alpha_1 \mathbf{y}_1 + \alpha_2 \mathbf{y}_2) = \alpha_1 f_{objective}(\mathbf{y}_1) + \alpha_2 f_{objective}(\mathbf{y}_2) \tag{2.6}$$

These types of linear problems can be solved using Least Squares or Linear Programming [Boyd04]. The more generalized case occurs when the equality of Eq.2.6 is replaced by an inequality that leads to convex optimization problems useful for solving unimodal

problems. Convex problems can be solved using gradient based methods and simple heuristics for steepest descent. For nonlinear multimodal problems, gradients can lead to local solutions that are far from optimal.

Even though there are some specialized techniques that work under strict conditions, there are no standard approaches for nonlinear multimodal problems [Boyd04], so heuristics are often employed to reduce the parameter space and guide the search towards an optimal solution [EbSh07]. One class of heuristics is evolutionary algorithms that start with a random solution and iteratively improve the solution through evolutionary means modelled from nature (i.e., mutation, crossover, and social interactions) as shown in Fig. 2.4 [Holl75] [KeEb01]. The improvements are achieved through a selection process that bias the survivability of many good solutions while allowing a small portion of poor solutions to survive for the purpose of exploring the parameter space [EbSh07]. The balance between good and poor solutions is randomized to ensure there is a diverse population that can avoid or escape many local solutions while converging towards a goal [FlMa08]. Therefore, evolutionary optimization algorithms are well suited for ill-defined problems where new states can emerge within the application in which the algorithm is deployed.

The behaviour of an EA relates to the methodology for selecting new solutions to evaluate and guide the search towards the solution. For example, non-evolutionary approaches such as a random search would not have any correlation between the various steps, while evolutionary processes expect long-term correlations that indicate the system is slowly moving towards an objective. The long-term correlation does not imply a steepest-descent, but rather on average, the solutions move in a certain direction while maintaining some level of randomness to explore the parameter space. Furthermore, the correlation is not fixed in time, but rather it can change over time as new information becomes available and produces emergent behaviours in the algorithm.

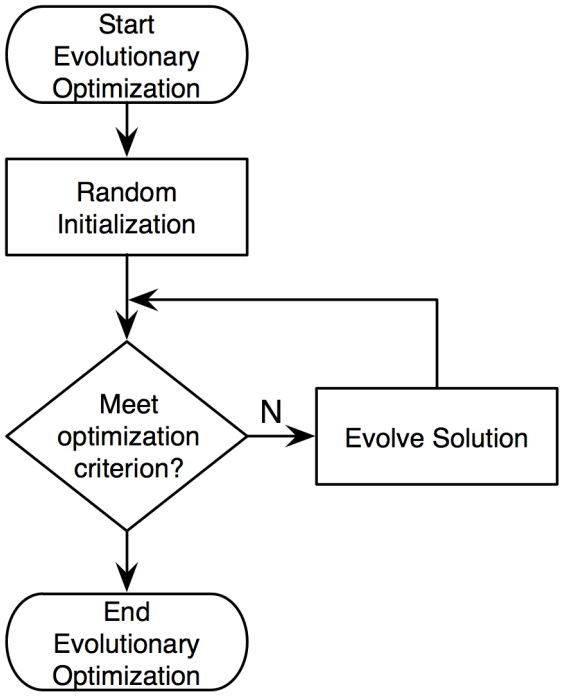


Fig. 2.4: Structure of a typical evolutionary optimization algorithm [ScKi11a].

The following sections provide a brief description of PSO and some related algorithms. Comments on the merits of different algorithms are presented as they relate to the core application of this thesis. Some general observations on the behaviour of each algorithm are also provided for the purpose of selecting the appropriate algorithm for this thesis.

2.2.1 Simulated Annealing

Simulated annealing (SA) is modelled by the annealing process from material sciences, in which the malleability of metals decreases with temperature [Kirk83]. The algorithm commences with a random solution at a high synthetic temperature. A new candidate solution is generated at each iteration by perturbing the current solution. This candidate

solution is accepted if it is better than the current solution. However, if the candidate is worst than the current option, then it is accepted based on the Boltzman criterion shown in Eq. 2.7, thus allowing some exploration of the parameter space [Kirk83]. Then, like in the annealing process, the temperature is slowly lowered to harden the metal, thus reducing the number of bad candidate solutions that are accepted. This reduces the exploration of the parameter space as the algorithm converges towards a solution.

$$e^{-\Delta D/T_{SA}} > random(0,1) \tag{2.7}$$

where, ΔD is the distance between the current and candidate solutions, T_{SA} the temperature, and $random(0,1)$ a uniformly selected random number between 0 and 1 to accept or reject candidate solutions.

The algorithm does not define either the encoding method or perturbation method, therefore making SA suitable for both discrete and continuous problems without any major modifications. This property is not common of all evolutionary algorithms and thus many methods for transforming solutions are often employed to make naturally continuous algorithms work with discrete or combinatorial problems and vice-versa.

The balance between exploration and convergence in SA is governed by the temperature for T_{SA}. Steep gradients are susceptible to getting trapped in local solutions and can often behave like greedy algorithms where only better solutions are accepted. On the other hand, very slow gradients can explore large areas of the parameter space, take longer to converge, and still not guarantee an optimal solution. Therefore, the temperature gradient is often tuned to the problem through the use of linear parameters or nonlinear parameters like logarithmic curves. Further improvements to the algorithm include the use of chaotic trajectories for candidate solutions to avoid repeating the same sequence and therefore accelerating the convergence [ShKi96].

The behaviour of SA can be extracted from the sequence of solutions generated and evaluated. The solution evaluated creates a trajectory as it moves through the parameter space. The domain for generating solutions diminishes over time as does the rate of accepting large perturbations to explore the parameter space. The process does not provide feedback to help generate new solutions that would improve on the current iteration results.

2.2.2 Genetic Algorithms

Genetic algorithm (GA) is analogous to the evolution of organisms where the genetic codes can improve over many generations [Holl75]. In this class of algorithms, a candidate solution is known as a chromosome and consists of genes representing the different parameters being optimized. A population of chromosomes evolve in the search for a solution by modifying the genetic codes via crossover and mutation [Fogo97]. A portion of the population (usually half) advances to the next generation based on a random selection in which the probability of an individual being selected is proportional to its fitness [EbSh07]. The surviving chromosomes serve as parents for a new generation of candidate solutions. Crossover produces the new solutions, known as children, by combining elements of the genetic code from two parent chromosomes. A stochastic process is used to select a very small percentage of genes to randomly modify through mutation to stimulate the population and prevent it from getting trapped in local minima.

The encoding process in GA is commonly performed by concatenating all the genes into a single string like schemata theory [Holl75] [EbSh07]. This method may produce invalid solutions that fall outside the predefined range of values for a given solution, thus special consideration must be paid to the crossover and mutation processes.

There are many extensions or variations to the classical GA that can speed convergence.

For example, *elitist strategies* are often used to ensure that a small percentage of the highest ranked chromosomes are advanced to the next generation in order to remove the stochastic nature of the roulette that can on occasion remove a good candidate solution from the population. Furthermore, there are many different types of crossover mechanisms, like: *one-point crossover*, where a single point is pre-selected; thus, the child receives the front half of the genes from one parent and the bottom half from the other [EbSh07] or *uniform crossover* where the point for the one-point crossover is randomly selected for each set of parents.

Some variations of crossover perturbations in a GA can retain elements of previous solutions that reveal some correlations over successive iterations. The challenge is that at each iteration, half the population ceases and new children emerge, thus generating a tree where half the branches at each level fail to recreate. Furthermore, a correlation can be established for some crossover/encoding pairs where the procedure helps narrow down the search for multiple parameters at the same time (i.e., the crossover does not improve one parameter at the expense of another). This is an improvement over SA because the crossover and new generations provides the necessary feedback (through memory retention) to improve solutions.

2.2.3 Ant Colony Optimization

The *ant colony optimization* (ACO) is modelled after the behaviour of ants searching for food by communicating with their peers through the use of scent chemicals called pheromones [DoMC96]. The use of chemical messengers to preserve good solutions serves to guide the search over time similar to the retention of high fitness genetic codes in GAs. In the algorithm, the ants are provided with a finite set of possible paths that connect them to the food source, and their objective is to find the shortest path. During the first iteration,

of the algorithm the ants travel through the graph and find the food source while depositing pheromones along the way. In subsequent iterations, the ants repeat the process and use the previously available information to select routes whenever they reach a vertex. The selection is based on the *probability mass function* (pmf) given by each route's pheromone concentration and desirability, where the desirability is generally inversly proportional to the length of the edge in the ants path. Over time, the pheromones evaporate, thus reducing the chances that ants travel through non-optimal paths that lead to local solutions. After many iterations, the majority of the ants travel through the optimal paths and thus trigger the termination criterion for the algorithm.

Because of the underlying graph metaphor in the algorithm, it is ideally suited to solve problems like the traveling salesman or similar combinatorial challenges because of the predefined and finite paths available [Soch08] [SoDo08] [Math00]. To extend the problem to the continuous domain, one can expand the pmf developed from pheromone concentrations in predefined paths to a *probability density function* (pdf) defining the volumetric element in which the scent from the chemical can be detected [Soch08]. The ants then move towards the food source by selecting steps from a pdf that take into account the overall structure of the pheromone distribution. From an implementation perspective, the number of possible solutions stored (i.e., pheromone pdf definitions) can grow very quickly, so the evaporation process keeps a table with the last few generations worth of solutions, thus reducing the memory requirements for the algorithm [Soch08].

At the end of each iteration, the trajectory of the ants forms one possible solution that is evolved over time. Tracing the possible solution serves to define the behaviour of the algorithm. Like in SA, there is a single trajectory of solutions, but with a feedback behaviour that can expose more long-term correlations. Unlike GA, the paths of individual ants are not important, but rather the path found by the colony form the trajectory that define the behaviour.

2.2.4 Particle Swarm Optimization

The PSO algorithm was developed by Kennedy and Eberhart based on the interactions of organisms as seen in flocks of birds and schools of fish [KeEb95]. In this model, organisms are represented by S_K massless particles that traverse a D-dimensional parameter space looking for a solution. At each iteration, the particles select their next position based on a weighted sum combining the best solution found by the particle and the neighbourhood. Details on PSO are provided in Ch. 3.

The behaviour of the algorithm may be characterized by the trajectory of the particles as they converge on a solution. A single particle moves in ways similar to the temporary solutions in SA, however, it is greedy about storing its previous best position instead of using a probability to accept solutions. Both GAs and PSO produce multiple solutions at each iteration. The difference is that in GAs a portion of the solutions are replaced at each iteration by new offspring, while in PSO the same set of particles is used for the entire duration of the algorithm run. In the original implementation, all the particles collaborate together from start-to-finish; thus, their trajectories towards a solution can be compared to establish trends that help guide particles towards the solution [ScKi11a]. More details on characterizing the behaviour of PSO are provided in Ch. 3 and Ch. 4.

2.2.5 Other evolutionary algorithms

In addition to the aforementioned evolutionary algorithms, there are others like *taboo search*, *bees algorithm*, and *memetic algorithms* that can also produce good results in many applications. *Taboo search* is specifically designed to improve the performance of discrete local searches by preventing the algorithm from getting stuck in local solutions, recently visited solutions, or plateaus in the parameter space where many solutions have the same fitness [Glov89] [Glov90]. The algorithm keeps a fixed sized circular list of recently visited

solutions that are considered "taboo" for the next set of iterations until those conditions expire. The *bees algorithm* mimics the foraging behaviour of honey bees to find a solution. In this approach, the algorithm keeps track of the best patches found by the colony of bees and assigns bees to the sites for local searches. The assignments are based on the fitness of each patch and the best fitness found is entered into the new patch list. To prevent getting stuck in local solutions based on the initial random assignments, a subset of bees is always used to randomly explore the parameter space looking for new patches. Finally, *memetic algorithms* are based on Richard Dawkins' idea of *meme* that extended the evolutionary notions with culture [FlMa08]. This is often reproduced as a hybrid algorithm that combines elements of evolution (i.e., genetic algorithms) with local search refinements based on individual learning procedures.

2.3 Summary

This chapter highlighted some of the fundamental principles behind scheduling in real-time systems and evolutionary algorithms. The brief descriptions that characterize the behaviour of evolutionary algorithms show vibrant research areas that can be exploited to improve the performance of these algorithms for specific applications such as scheduling. Chapter 3 focuses on the PSO algorithm in order to develop an intuitive understanding of the parameters that can be used in an in-depth analysis of the behaviour such that it can solve difficult problems as the one proposed in Ch. 4.

Chapter 3

Theoretical Background on Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm was introduced by James Kennedy and Russell Eberhart in 1995 after extensive work simulating the behaviour of groups of organisms [EbKe95]. The algorithm combines elements of artificial intelligence and evolutionary programming to solve continuous, nonlinear, unconstrained optimization problems [KeEb01]. Since its inception, there have been many proposed improvements to the algorithm. This chapter provides an overview of the original PSO algorithm and some of the significant improvements commonly employed.

3.1 The Original Algorithm

The concept of PSO originated from an attempt to model the behaviour of flocks of birds, school of fish, and swarming theory, particularly focusing on the movements of large groups and the collaboration through information sharing that helps the organisms work

together towards some common goal [KeEb95]. Early attempts to model the behaviour of swarms focused on the local processes within each organism that determined the behaviour. For example, the relative position of other members of the group that would maintain the optimum distance between neighbours [KeEb01]. These studies were later augmented to incorporate information sharing between organisms as suggested by evolutionary theory [KeEb01]. By sharing information, new global processes emerged and the dynamics of the swarm began to replicate those of living organisms [John01].

The general idea behind PSO is a search by a colony of particles moving through a parameter space representing elements of the real world. The movements of the volume-less particles are derived from their previous position and a weighted sum of the best states for the particle and the collective knowledge of the particle's neighbourhood as stated in Eq. 3.1.

$$next\_state = curr\_state + personal\_influence + social\_influence \tag{3.1}$$

The *state* of a particle is given by its position and velocity during one discrete time iteration. The iterations are equally spaced time intervals. Thus, the *next_state* refers to the new state of the particle, while *curr_state* is the state during the previous iteration. The *personal_influence* is the memory of the particle's previous best potion. While the *social_influence* refers to the memory of the best state found by the particle's neighbourhood.

In essence, the actions of each particle are guided by a combination of the current state, their own beliefs, and those of the neighbourhood. Just like in the case of humans, the balance between the personal and social beliefs varies over time allowing particles to explore the parameter space while also following best trends accepted by the community in order to evolve towards a solution to the problem [KeEb01].

3.1.1 Mathematical Realization

A more detailed version of Eq. 3.1 is shown in Eq. 3.2 [KeEb95]. Using bold notation for vectors and regular font for scalars, the state for each particle, *curr_state* and *prev_state*, shown in Eq. 3.1 are defined by the position, $\mathbf{S}_x$, and velocity, $\mathbf{S}_v$, respectively. The vectors represent the D-dimensions in the parameter space where the particles are moving. $\mathbf{S}_{x,k}(n)$ refers to the position of the k^{th} particle at iteration n. The *personal_influence* is calculated as the difference between the current position and the previous best position, $\mathbf{S}_p$, for that particle and multiplied by a weight, $S_{\varphi 1}$, and a uniformly distributed random number between 0 and 1. Similarly, the *social_influence* is calculated as the difference between the current position and the previous best position for the neighbourhood (represented by the g^{th} particle) of the particle and then multiplied by a weight, $S_{\varphi 2}$, and a uniformly distributed random number between 0 and 1. The equations for $\mathbf{S}_p$ and S_g are defined in Sec. 3.1.2.2. The original PSO defined the neighbourhood as the set of all particles, thus guiding the particle's movements based on their own personal experience and the best solutions from the global community. The independent $S_{\varphi 1}$ and $S_{\varphi 2}$ can either amplify or reduce the weighted sum influences in different ways, and can impact the convergence of the algorithm (see Sec. 3.2.3) [KeEb01].

$$\begin{aligned}\mathbf{S}_{v,k}(n) = {} & \mathbf{S}_{v,k}(n-1) + \\ & S_{\varphi 1} \times random(0,1) \times (\mathbf{S}_{p,k} - \mathbf{S}_{x,k}(n-1)) + \\ & S_{\varphi 2} \times random(0,1) \times (\mathbf{S}_{p,g} - \mathbf{S}_{x,k}(n-1))\end{aligned} \tag{3.2a}$$

$$\mathbf{S}_{x,k}(n) = \mathbf{S}_{x,k}(n-1) + \mathbf{S}_{v,k}(n) \tag{3.2b}$$

The perturbations or new positions of a particle in its trajectory may be determined by iterating Eq. 3.2. The addition of the previous state provides a positive feedback that

serves to increase the speed of particles traveling in the same direction for multiple iterations. This behaviour is commonly observed in the early stages of the algorithm when the random initialization places a particle far away from a possible good solution causing particles to continue moving in the current direction in large steps, and converging to a solution faster than if the previous state was not included. As a side consequence of this type of accelerated movement, a particle may overshoot the target, and continue doing so, in an oscillatory manner [Kenn97]. These movements depend on the shape of the parameter space around the solution.

Furthermore, depending on the parameter space, the accelerated movement may lead to some particles becoming unstable and leaving the region of the parameter space under consideration and escaping the predefined area for the parameter space. The randomization of $S_{\varphi 1}$ and $S_{\varphi 2}$ helps to reduce the possibility of overshoots by placing more/less importance on the personal or social influences at each iteration. More insight into the behaviour of the particles when they overshoot the target is provided in Sec. 3.1.4 through different visualizations of the algorithm.

Solving these equations iteratively for all particles determines the movement of the particles towards a solution in the parameter space. As particles converge on a solution, it is possible that they may overlap the same location. This is acceptable and in some cases a good indicator of convergence as described in Sec. 3.2.5.

3.1.2 The PSO Algorithm

The PSO algorithm is divided into the initialization and the main algorithm as described by Algorithms 3.1 and 3.2 respectively.

3.1.2.1 Algorithm Initialization

In the initialization stage, the position and velocity of each particle are generated randomly for each dimension of the problem, as shown in Algorithm 3.1. The initialization is carried out for each of the D-dimensions of the position, $\mathbf{S}_x$, velocity, $\mathbf{S}_v$, and previous best, $\mathbf{S}_p$ vectors. The best position for each particle is initialized to the particle's position, thus the first iteration the algorithm will use social influences only.

Algorithm 3.1 PSO Initialization

1: **for** $S_k = 1$ to S_K **do**
2: **for** $d = 1$ to D **do**
3: $S_{x,k,d}(n) \leftarrow random(S_{Xmin}, S_{Xmax})$
4: $S_{v,k,d}(n) \leftarrow random(S_{Vmin}, S_{Vmax})$
5: $S_{p,k,d}(n) \leftarrow S_{x,k,d}(n)$
6: **end for**
7: **end for**

The position is selected from a uniform distribution ranging from S_{Xmin} to S_{Xmax}. The uniformity ensures that the locations of the particles are evenly distributed in the parameter space. In constrained applications, the position extrema are determined from physical constraints in the problem, thus preventing some invalid solutions from being selected during the initialization. If there are no constraints, the parameters are estimated from simulations such that the initial distribution of particles covers a wide range of solutions to reduce the likelihood of premature convergence. This can be narrowed down with *a priori* knowledge of the likely range of values for the solution, thus reducing the length of the search for the particles [WoMa97].

The early implementations of the PSO algorithm defined the initialization position extrema as given by Eq. 3.3, thus defining a symmetric range of values. However, as noted by

[FoBe95], this decision can bias the movement of the particles for some test functions.

$$S_{Xmin} = -S_{Xmax} \tag{3.3}$$

In contrast, the velocity extrema play a significant role in the original PSO and, therefore, must be selected to large enough to escape local solutions in the parameter space while also limiting the step size to prevent particles from escaping the predefined area of the parameter space. A maximum step size of $\frac{1}{2}(S_{Xmax} - S_{Xmin})$ is typically selected to force particles to traverse and explore when moving across the parameter space [KeEb01]. Larger ranges are discouraged even though they can speed up convergence because skipping through large areas can lead to premature local solutions. Limiting the step size significantly, as shown in [KeEb01] and [ScKi11a], slows down the algorithm while retaining the same behaviour. Since small step sizes can get stuck in local solutions more easily, this is only performed in benchmark scenarios where the maximum velocity is known to be large enough to overcome local solutions.

Finally, the best position for each particle, $\mathbf{S}_p$, is set to the particle's position, thus making the first step the result of neighbourhood influences only. Although randomly selecting the values this vector do not hurt the performance of the algorithm, most authors opt to let the particles explore their own paths to match the behaviour of the swarms being modelled (i.e., [KeEb01]).

3.1.2.2 Main Algorithm Loop

The main loop for the PSO algorithm is shown in Algorithm 3.2. At each iteration, the algorithm updates the state of all S_K particles. The update consists of three steps; to update the personal memory (Lines 3–5); find the best particle in the neighbourhood

(Line 6); and then update the position/velocity vectors (Lines 8–10).

Algorithm 3.2 PSO Algorithm

```
1: repeat
2:   for S_k = 1 to S_K do
3:     if G(S_{x,k}) < G(S_{p,k}) then
4:       S_{p,k} ← S_{x,k}
5:     end if
6:     S_g ← index of best neighbour for S_k^{th} particle
7:     for d = 1 to D do
8:       S_{v,k,d}(n) ← Result of Eq. 3.2a
9:       S_{v,k,d}(n) ∈ (S_{Vmin}, +S_{Vmax})
10:      S_{x,k,d}(n) ← Result of Eq. 3.2b
11:    end for
12:  end for
13: until termination criteria is met
14: return p_{best}
```

The memory update consists of evaluating the cost function, $G(\bullet)$ for both the current position, $\mathbf{S}_x$, and the best position found, $\mathbf{S}_p$. If the current position yields a better fitness rating, then it becomes the current best position and the memory is updated. Thus, the particle's previous best position is determined through a greedy decision that is based only on the past best solutions, and no other global information is considered. While such greedy approach was not acceptable in SA, a Boltzman-like criterion is not needed in PSO because there are S_K solutions being stored in the swarm (one in each particle). This collection of previous solutions form the social influences overcomes the local minima traps that limit a non-social algorithm like SA.

Line 6 uses a linear search through all the particles in the neighbourhood to find the index with the best fitness that can guide the social component of Eq. 3.2. In the original PSO the neighbourhood was defined as the entire set, but other topologies are often used as outlined in Sec. 3.2.4. The linear search is a computationally expensive operation that evaluates the cost function for each particle in the neighbourhood. To avoid duplicating the calculations, some authors choose to cache the results by maintaining two additional vectors

for the fitness of each particle's current position, $\mathbf{S}_{fx}$, and the fitness of each particle's best position. $\mathbf{S}_{fp}$. The vectors are only updated when $\mathbf{S}_x$ or $\mathbf{S}_p$ are updated. The fitness vector would be initialized in Algorithm 3.1 after iterating through every dimension for a given particle. The vector is updated when a new position is committed for a particle after computing Eq. 3.2. Thus, all the comparisons of the particle's information are reduced to very simple operations. This implementation is shown along its algorithm complexity in Table 3.2.

The perturbations to the current position that lead to the evolutionary processes of the algorithm are defined in Lines 8–10. For each dimension, d, the parameters are updated to obtain a velocity vector that corresponds to the step size for the particle. As outlined in Sec. 3.2.2, the velocity extrema are used to prevent the step size from growing too large as this can cause particles to become unstable.

The update equation are often implemented as an *atomic commit* to allow all the changes to take effect in a single operation after the loop betweens Lines 2–12 completes. This maintains the fidelity of the swarm behaviour being simulated because all particles would move at the same time and have the same information available to them for that iteration. Failing to do this, the implementation can bias the movement of some particles based on their index into the population data structure. Practically, this is accomplished by storing the new positions in Line 10 in a temporary position vector and then copying those values before the next iteration of the algorithm. Note, that when using a global topology (as described in Sec. 3.2.4), it is possible to move that operation outside of the particle loop after the atomic commit. This removes any duplicated computations, however it does not allow different topologies to be tested.

The termination criteria for the algorithm is met if either a maximum number of iterations is reached or a special condition is met [ZiPL05]. The maximum number of iterations

is often selected in real-time applications as it helps enforce the timing requirements for the system. The special conditions are application dependant.

3.1.3 PSO Algorithmic Complexity

The algorithm complexity is seldom discussed in evolutionary algorithms because many of the parameters are dependant on the problem being evaluated. The value of identifying some general bounds for the algorithm are important to gauge the applicability to specific problems, such as real-time scheduling.

The algorithmic complexity of the original PSO without any special implementation optimizations is $\mathcal{O}(N_{max}S_K^2D^2)$ as shown in Table 3.1. For this estimate, the three key assumptions were (i) the use of a global topology for a particle's neighbourhood, (ii) that the cost function requires traversing through the entire position vector to compute the fitness, and (iii) the termination criterion is a fixed maximum number of iterations, N_{max}. One of the most costly operations in this case is the computation of every particle's fitness in a global topology each time the new best index in the neighbourhood, S_g, is computed in Line 13. This seemingly simple operation requires computing the fitness and comparing the S_k^{th} particle to each other particle in the set to identify the neighbourhood best, S_g. In contrast, changing to a local neighbourhood configuration with S_n neighbours can reduce the complexity to $\mathcal{O}(N_{max}S_KD^2S_n)$, where $S_n < S_K$ and the associated fitness computations are fewer than the D^2 perceived from the complexity measure.

The complexity can be reduced to $\mathcal{O}(N_{max}S_KD)$ through a smart implementation that addresses the key computationally expensive portions of the algorithm. The main changes are shown in Table 3.2. In essence, this implementation removes any duplicate calculations through dynamic programming that caches (i) the fitness of each particle, (ii) the fitness of each particle's best position, and (iii) and the index of the best particle in the

Table 3.1: Algorithmic complexity of PSO

	Line of program	Comment	Complexity
1:	**for** $S_k = 1$ to S_K **do**	*Initialization*	
2:	**for** $d = 1$ to D **do**		S_K
3:	$S_{x,k,d}(n) \leftarrow random(S_{Xmin}, S_{Xmax})$		$S_K D$
4:	$S_{v,k,d}(n) \leftarrow random(S_{Vmin}, S_{Vmax})$		$S_K D$
5:	$S_{p,k,d}(n) \leftarrow S_{x,k,d}(n)$		$S_K D$
6:	**end for**		
7:	**end for**		
8:	**for** $n = 1$ to N_{max} **do**	*Main loop*	
9:	**for** $S_k = 1$ to S_K **do**	*Initialization*	N_{max}
10:	**if** $G(\mathbf{S}_{x,k}) < G(\mathbf{S}_{p,k})$ **then**		$N_{max} S_K$
11:	$\mathbf{S}_{p,k} \leftarrow \mathbf{S}_{x,k}$	*Copy vector*	$N_{max} S_K D$
12:	**end if**		
13:	$S_g \leftarrow$ index of best neighbour	*Global topology*	$N_{max} S_K^2 2D^2$
14:	**for** $d = 1$ to D **do**	*Update particles*	
15:	$S_{v,k,d}(n) \leftarrow$ Result of Eq. 3.2a		$N_{max} S_K D$
16:	$S_{v,k,d}(n) \in (S_{Vmin}, +S_{Vmax})$		$N_{max} S_K D$
17:	$S_{x,k,d}(n) \leftarrow$ Result of Eq. 3.2b		$N_{max} S_K D$
18:	**end for**		
19:	**end for**		
20:	**end for**		
21:	**return** $\mathbf{S}_{p,best}$		$S_K 2D^2$

neighbourhood. Furthermore, this implementation introduces a new variable S_{gnext} that pre-computes what S_g should be in the next iteration when a global topology is assumed. Although it is possible to incorporate the cost function directly into the algorithm and reduce a few loops (i.e., pre-compute line 9 within the loop in Lines 4–8), it is common to leave the cost function separately to be able to adjust it for a particular application.

Having optimized the implementation to accelerate PSO performance, the $\mathcal{O}(N_{max}S_K D)$ is dependant on three key parameters. The number of iterations, N_{max}, is often set as a worst-case scenario for the runtime of the algorithm in case a primary termination criterion fails to find a solution. This acts like a watchdog timer that timeouts after a predetermined length of time to prevent the algorithm from an endless search. This can then trigger the algorithm to be re-initialized or the use of the best solution available at that time. The number of particles, S_K, is discussed in Sec. 3.2.1 as it not only affects the computational complexity of PSO, but also the exploration of the parameter space that leads to finding good solutions. Finally, the dimensionality of the problem is often characterized by the number of independent variables in the problem.

Table 3.2: Algorithmic complexity of PSO with an array of cached fitness computations and pre-computed best index in neighbourhoods.

	Line of program	Comment	Complexity
1:	$S_g \leftarrow 1$	*Initialize* S_g	
2:	$S_{gnext} \leftarrow 1$		
3:	**for** $S_k = 1$ to S_K **do**	*Initialization*	
4:	**for** $d = 1$ to D **do**		S_K
5:	$S_{x,k,d}(n) \leftarrow random(S_{Xmin}, S_{Xmax})$		$S_K D$
6:	$S_{x,k,d}(n) \leftarrow random(S_{Vmin}, S_{Vmax})$		$S_K D$
7:	$S_{p,k,d}(n) \leftarrow S_{x,k,d}(n)$		$S_K D$
8:	**end for**		
9:	$S_{fx,k} \leftarrow G(\mathbf{S}_{x,k})$	*Cache fitness*	$S_K D$
10:	$S_{fp,k} \leftarrow S_{fx,k}$		S_K
11:	**if** $S_{fp,k} < S_{fg,k}$ **then**		S_K
12:	$S_{gnext} \leftarrow k$	*Cache* S_g	S_K
13:	**end if**		S_K
14:	**end for**		
15:	**for** $n = 1$ to N_{max} **do**	*Main loop*	
16:	$g \leftarrow g_{next}$		N_{max}
17:	**for** $S_k = 1$ to S_K **do**	*Initialization*	
18:	**if** $S_{fx,k} < S_{fp,k}$ **then**	*Cached fitness*	$N_{max} S_K$
19:	$\mathbf{S}_{p,k} \leftarrow \mathbf{S}_{x,k}$	*Copy vector*	$N_{max} S_K D$
20:	$S_{fp,k} \leftarrow S_{fx,k}$		$N_{max} S_K$
21:	**end if**		
22:	**for** $d = 1$ to D **do**	*Update particles*	
23:	$S_{v,k,d}(n) \leftarrow$ Result of Eq. 3.2a		$N_{max} S_K D$
24:	$S_{v,k,d}(n) \in (S_{Vmin}, +S_{Vmax})$		$N_{max} S_K D$
25:	$S_{xtemp,k,d}(n) \leftarrow$ Result of Eq. 3.2b	*Temporary state*	$N_{max} S_K D$
26:	**end for**		
27:	**end for**		
28:	**for** $S_k = 1$ to S_K **do**		
29:	$\mathbf{S}_{x,k} \leftarrow \mathbf{S}_{xtemp,k}$	*Copy vector*	$N_{max} S_K D$
30:	$S_{fx,k} \leftarrow G(\mathbf{S}_{x,k})$	*Cache new fitness*	$N_{max} S_K D$
31:	**if** $S_{fp,k} < S_{gnext,k}$ **then**		$N_{max} S_K$
32:	$S_{gnext} \leftarrow k$	*Cache* S_g	$N_{max} S_K$
33:	**end if**		$N_{max} S_K$
34:	**end for**		
35:	**return** $\mathbf{S}_{p,gnext}$		1

3.1.4 Visualization

One of the most powerful techniques for studying the PSO algorithm is through visualizing the movement of particles through the parameter space, their interactions, and how they move towards a solution [Kenn97]. The most common technique is common to all evolutionary algorithms and consists of plotting the fitness of the best solution found versus the algorithm iteration as shown for an arbitrary particle ($S_k = 7$) in Fig. 3.1 and Fig. 3.2.

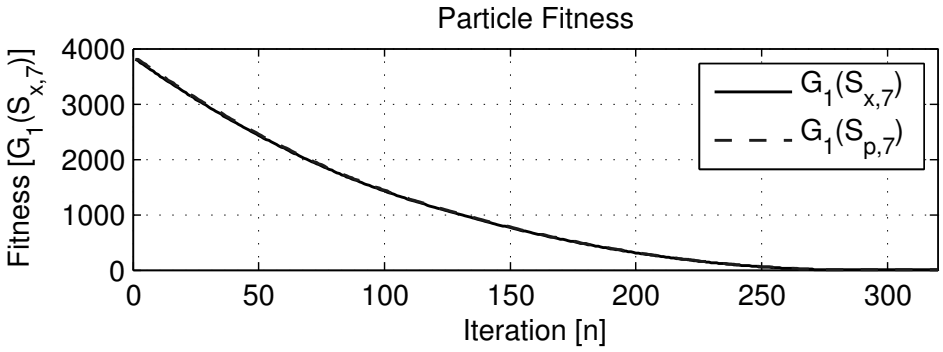


Fig. 3.1: The fitness for the current and best solutions found by the particle are almost identical at every iteration. This is a unique feature that happens in smooth, concave, unimodal functions like the Sphere.

The fitness plot shows the rate at which the algorithm approaches a solution and can provide some guidance for selecting parameters based on the convergence rate of the solution. The main limitation with these plots is that they fail to provide a full understanding of the behaviour of PSO. As such, it is not always clear what the effect is from different parameters and what behaviours are simply a result of the stochastic nature of the algorithm. For example, Fig. 3.2 shows a plateau for the best solution found by the particle around iterations 12-50 which can be a result of many things such as getting temporarily stuck in a local solution or simply failing to find a better solution.

To overcome the limitations of fitness plots, there are unique visualization techniques for

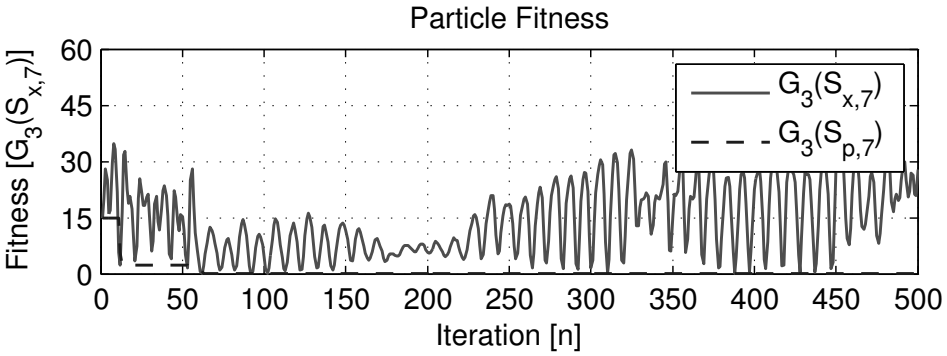
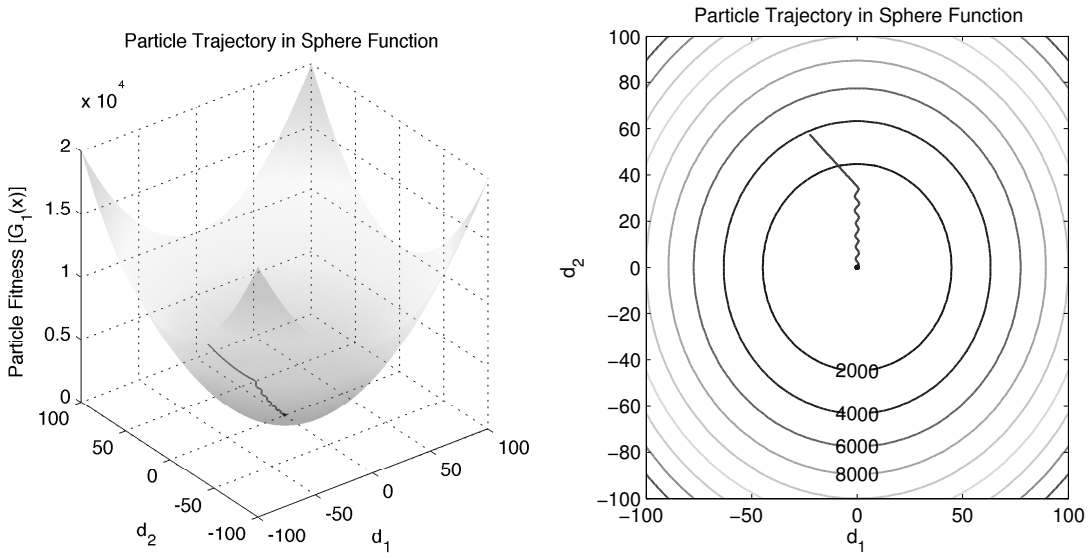


Fig. 3.2: Fitness plot for $S_k = 7$ in PSO on Rastrigin function. The fitness of the current position varies greatly at each iteration and may continue for many iterations. In contrast, the fitness of the best solution found by the particle is a non-increasing curve with plateaus while the particle explores the parameter space.

some EAs. For example, in GAs, many authors choose to represent each chromosome as a row in a table with different colours used for each gene. The result shows how predominant patterns emerge in the population for strong evolutionary traits, like playing the game Mastermind to find a sequence that breaks the code [KaCa03]. In the case of PSO, there are two techniques commonly used to study the behaviour of the algorithm for low dimensional problems where $D \leq 2$: using a 3D plot or a contour map, as shown in Fig. 3.3 and Fig. 3.4, for the Sphere and Rastrigin functions respectively. The same work can be extended to live updating computer graphics or videos to help visualize the full swarm without congestion from the entire trajectory. These representations show two dimensions along the x-axis and y-axis and use the z-axis (or projected contour map lines) to depict the different fitness values. Plotting the particles' movement reveals overshoots in trajectories that cause the particle to oscillate and circle towards the solution. This behaviour instilled the need for damping factors that prevent oscillations and accelerate convergence [Kenn97].



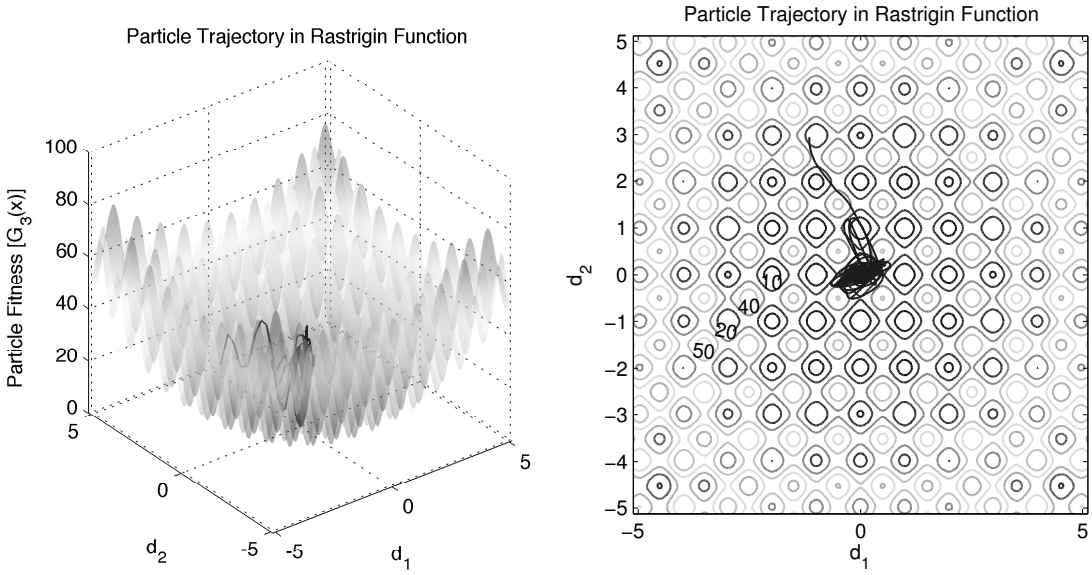
(a) 3D representation on $G_1(\mathbf{S}_{x,7})$ [ScKi11a]. (b) Contour projection on $G_1(\mathbf{S}_{x,7})$.

Fig. 3.3: 3D and contour representations for a single particle ($S_k = 7$) trajectory in PSO on Sphere function. (a) Shows the 3D representation with the Sphere function providing a reference point for the trajectory. (b) The contour map projection that makes it easier to identify the trajectory of the particle. The particle starts at $S_{x,7} = \{-22.5, 57.5\}$ and initially travels in a diagonal to optimize one dimension before oscillating towards the global optimum. The size of the oscillations decreases over time showing the particle is slowly converging.

As informative as these plots are, they are limited to two dimensions and require a fully defined parameter space (e.g., Sphere or Griewank functions) that defeats the overall purpose of the algorithms. Therefore, these techniques are reserved for educational purposes and verifying algorithm implementations. A novel particle trajectory representation that overcomes these limitations is presented in Sec. 4.1.4.

3.2 Variations and Comparisons

There are many modifications, improvements, and variations to the original PSO algorithm. In recent years, researchers have attempted to define a standard for PSO based on



(a) 3D representation on $G_3(\mathbf{S}_{x,7})$ [ScKi11a].

(b) Contour projection on $G_3(\mathbf{S}_{x,7})$

Fig. 3.4: 3D and contour representations for a single particle ($S_k = 7$) trajectory in PSO on Rastrigin function. (a) Shows the 3D representation with the Rastrigin function providing a reference point for the trajectory. (b) The contour map projection that makes it easier to identify the trajectory of the particle. The particle starts at $S_{x,7} = \{-1.1, 2.9\}$ and initially travels in a diagonal to optimize one dimension before oscillating towards the global optimum. Note that the particle appears to explore the parameter space around {0, 1} thus matching the increased fitness shown in Fig. 3.2.

important contributions described in the literature [BrKe07]. In practice, many researchers choose to work with the original algorithm and known parameters in order to verify their implementations against existing literature, before applying it to their specific problem. This section describes some significant contributions, parameters selected, and comparisons to the original algorithm.

3.2.1 Population Size

The number of particles, S_K, has a great influence on the efficiency of PSO as this parameter directly affects the computational complexity and memory requirements as the

algorithm must evaluate the position of each particle at each iteration, as described in Sec. 3.1.3. Intuitively, one can assume that more particles implies a greater exploration of the parameter space that can make the algorithm more robust against local minima. Unfortunately, that is not always the case, as some topologies (see Sec. 3.2.4) can help in the exploration without increasing S_K [BrKe07].

In the original algorithm, Kennedy and Eberhart suggest that anywhere from 15 to 30 particles are sufficient to solve classical test functions for evolutionary problems as defined in Sec. 4.1.3 [KeEb95]; in contrast many authors prefer to use larger populations in unknown parameter spaces to explore more candidate solutions. To evaluate the sensitivity to S_K, Trelea [Trel03] and Angeline [Ange98] performed independent tests using $S_K = \{15, 30, 60\}$ and $S_K = \{125\}$ particles respectively with fixed weights and monitored the number of iterations until the different swarms reached a similar quality of solutions. These show that increasing S_K speeds the performance of the algorithm over some test functions. In essence, the empirical results do not provide a definitive value for the swarm size that is optimal across all problems, so some tuning of parameters is required for specific applications. In general, swarm sizes between $S_K = \{20 \text{ to } 100\}$ particles work on most problems [KeEb01] [BrKe07].

3.2.2 Limiting Step Size

The movement of the particles as per Eq. 3.2 uses a weighted difference between the particle's best and neighbourhood best, which can cause, under some situations, undesirable trajectories for some particles. For example, a system of $S_K - 1$ particles are initialized to be within a small distance from the optimal solution, while one particle is randomly initialized very far away from the rest of the swarm. In that case, the step size calculated for that the particle is very large and further increased through inertia weights, thus iteratively

overshooting the target and possibly leading into uncontrolled trajectories. There are a few different techniques to prevent these scenarios by dampening the oscillations through (i) maximum step sizes, (ii) inertia weights, and (iii) constriction coefficients as described in the following subsections.

The simplest method to control the velocity is to set hard limits, S_{Vmin} and S_{Vmax}, as shown in Line 9 of Algorithm 3.2. If the velocity (positive or negative) ever grows beyond these boundaries, the step size is cutoff to prevent an explosion in the particle trajectory [EbSh00]. This ensures that if a particle is far away from the neighbourhood best, S_g, and a high social coefficient, $S_{\varphi 2}$ is present, the steps do not grow larger in each step. Since there is no rule for selecting the value of S_{Vmax}, some authors use a value less than or equal to half the initial domain space as shown in Eq. 3.4 [KeEb01] [Berg01]. This allows the particle to traverse a large portion of the area in one iteration, while the overshoot is handled by the next iteration bringing the particle back towards the neighbourhood best. Using very small step sizes reduces the overshoots, but prolongs the algorithm as evaluated by [KeEb01]. The velocity is studied in Sec. 4.1.4 to analyze the trajectory of the algorithm for the purpose of characterizing the behaviour of PSO.

$$S_{Vmax} \leq \frac{S_{Xmax} - S_{Xmin}}{2} \tag{3.4a}$$

$$S_{Vmin} = -S_{Vmax} \tag{3.4b}$$

Another approach to limit the step size is to use an inertia weight, S_ω that produces a similar effect to the SA temperature gradient, T_{SA}, that reduces the step size over time [ShEb98]. This enables a global search during the initial phases of the algorithm and then decreases the step size to search locally around a solution. In practice, this is incorporated into the main algorithm by modifying the key equations for the movement of the algorithm

from Eq. 3.2 to those shown in Eq. 3.5, where the inertial movement carried over from the previous iteration is reduced over time. Extensive studies performed by Shi, Eberhart, and Kennedy [ShEb99] [KeEb01] revealed that it is sufficient to vary $S_\omega = S_{\omega,max} \rightarrow S_{\omega,min} = 0.9 \rightarrow 0.4$ over the duration of $S_{\omega,eph}$ iterations of the algorithm to decrease the speed of the particles. The decrements for S_ω can be either linear or nonlinear [Cook10]. The duration is problem dependant and often selected to a value greater than the expected average for the algorithm in order to allow the particles to escape local solutions. In general 1,500 iterations seemed acceptable for problems that normally converge in under 1,000 iterations (i.e., Sphere, Griewank) as shown by [ShEb98], [ShEb99], and [Berg01], while some authors extend this much more to 10,000 iterations to be able to handle more complex problems [Ange98]. Note that although using S_ω is considered a better approach for finding a solution, it is advisable to include a limiting S_{Vmax} as it is computationally inexpensive and helps prevent single particles diverging from the set.

$$\begin{aligned}\mathbf{S}_{v,k}(n) = {} & S_\omega \mathbf{S}_{v,k}(n-1) + \\ & S_{\varphi 1} \times random(0,1) \times (\mathbf{S}_{p,k} - \mathbf{S}_{x,k}(n-1)) + \\ & S_{\varphi 2} \times random(0,1) \times (\mathbf{S}_{p,g} - \mathbf{S}_{x,k}(n-1))\end{aligned} \tag{3.5a}$$

$$\mathbf{S}_{x,k}(n) = \mathbf{S}_{x,k}(n-1) + \mathbf{S}_{v,k}(n) \tag{3.5b}$$

The main limitation with the inertia weight is that it only compresses as the search converges, thus if along the way a better solution is found, the algorithm cannot adjust to expand the search and adapt to the terrain. To overcome this, Clerc and Kennedy simplified the model to analyze the convergence and introduced a set of five constriction coefficients that control the swarm's convergence tendencies for specific types of problems [ClKe02]. The most important of constriction coefficient is S_χ that is used to dampen

the oscillations by controlling the step size by replacing Eq. 3.2 with Eq. 3.6, where S_χ is given by Eq. 3.7 [ClKe02]. This parameter is derived by extracting the eigenvectors for the particle movement and finding the region for which the eigenvalues are real and negative to guarantee convergence. In this form, the condition $(S_{\varphi 1} + S_{\varphi 2}) > 4$ must be satisfied to guarantee convergence and prevent particles from traveling in a *spiral* around the solution [BrKe07]. Section 3.2.3 describes the personal and social weights in more depth and their effect of the behaviour of the particles and convergence of the algorithm.

$$\begin{aligned} \mathbf{S}_{v,k}(n) = {} & S_\chi\big(\mathbf{S}_{v,k}(n-1) + \\ & S_{\varphi 1} \times random(0,1) \times (\mathbf{S}_{p,k} - \mathbf{S}_{x,k}(n-1)) + \\ & S_{\varphi 2} \times random(0,1) \times (\mathbf{S}_{p,g} - \mathbf{S}_{x,k}(n-1))\big) \end{aligned} \tag{3.6a}$$

$$\mathbf{S}_{x,k}(n) = \mathbf{S}_{x,k}(n-1) + \mathbf{S}_{v,k}(n) \tag{3.6b}$$

$$S_\chi = \frac{2}{\left|2 - (S_{\varphi 1} + S_{\varphi 2}) - \sqrt{(S_{\varphi 1} + S_{\varphi 2})^2 - 4(S_{\varphi 1} + S_{\varphi 2})}\right|} \tag{3.7}$$

In order to compare these approaches, Eberhart and Shi tested the algorithm using S_ω and S_χ; they watched the particles fly through the projected parameter space through hundreds of runs searching for the optimum solution in six functions [EbSh00]. The analysis tested S_ω with the use of S_{Vmax} and compared it to the S_χ with a very high (and somewhat meaningless) S_{Vmax} and concluded that the constriction coefficient alone was not sufficient to bound the search causing the algorithm to take longer to converge to a solution. However, when comparing S_ω and S_χ both with S_{Vmax} as defined by the maximum value allowed in Eq. 3.4, the results showed that the performance of the constriction coefficient with a maximum step size improved significantly. Furthermore, Clerc's constriction method can be shown to be equivalent to the inertia weights when $S_\omega = 0.729$ and $S_{\varphi 1} = S_{\varphi 2} = 1.49445$

[EbSh00] [KeEb01] [Kenn07].

3.2.3 Social Influences

Conceptually, the random weights that balance the personal, $S_{\varphi 1}$, and social, $S_{\varphi 2}$, components in Eq. 3.2 should not be needed in optimization problems as the difference between the best and current positions should become increasingly positive or negative pulling the particle back in the corresponding direction [KeEb01]. These terms were initially added to help mimic the behaviours of real swarms where the particles have unique behaviours and do not always align perfectly at each step. Thus, in the original PSO implementation, the weights are multiplied by a uniformly distributed random number to vary the amount of exploration performed by the individual particle and how much it conforms to the behaviour of the swarm. In the original algorithm, these are set to $S_{\varphi 1} = S_{\varphi 2} = 2$ such that it would have a mean of 1 so that particles would overshoot the target about half the time and undershoot the target half the time [KeEb95]. Although it is not part of this thesis, other distributions for random numbers can produce good results. For example, normal distributions would encourage particles to take lots of small steps.

Practically, these weights play an important role into the behaviour of the swarm. To understand this effect, it is important to consider some extreme cases using simple functions, like the Sphere, as a benchmark with only S_{Vmax} to limit the step size. If $S_{\varphi 1} = S_{\varphi 2} = 0$, the trajectory of the particles increases linearly based on the initial velocity as $\mathbf{S}_{v,k}(n) = \mathbf{S}_{v,k}(n-1) + 0$. As the weights are increased between $0 < S_{\varphi 1}, S_{\varphi 2} \leq 1$, the trajectories for the particles begin to oscillate around a point with increasing frequency and reduced amplitude. Increasing the range $S_{\varphi 1}, S_{\varphi 2} > 1$ first shows the desired stochastic oscillations around the optimum and then begins to saturate when the multiplier is such that any small value maximizes the velocities at S_{Vmax} [OzMo99] [KeEb01].

For comparison, van den Bergh rewrote the problem as a recurrence relation with the inertia weight as shown in Eq. 3.8 [Berg01] [BeEn06]. Solving for the eigenvalues of the equation Eq. 3.9, revealed that the system would converge as long as $\max\{S_{\lambda 1}, S_{\lambda 2}\} < 1$. This occurs for a large range of combinations of S_ω, $S_{\varphi 1}$, and $S_{\varphi 2}$ [BeEn06]. Testing different combinations and plotting $0 < S_\omega \leq 1$ on the y-axis and $0 < S_{\varphi 1} + S_{\varphi 2} \leq 4$ on the x-axis showed that the system would converge for a large set of values within a slanted parabolic shape that included the set of parameters selected by Clerc described in Sec. 3.2.2.

$$\begin{aligned}\mathbf{S}_{x,k}(n+1) = (1 + S_\omega + S_{\varphi 1} \times random(0,1) + S_{\varphi 2} \times random(0,1)) \times \mathbf{S}_{x,k}(n) - \\ S_\omega \times \mathbf{S}_{x,k}(n-1) + S_{\varphi 1} \times \mathbf{S}_{p,k}(n) + S_{\varphi 2} \times \mathbf{S}_{p,g}(n)\end{aligned} \tag{3.8}$$

$$S_{\lambda 1} = \frac{S_\omega - S_{\varphi 1} - S_{\varphi 2} + \sqrt{(S_\omega - S_{\varphi 1} - S_{\varphi 2})^2 - 4\omega}}{2} \tag{3.9a}$$

$$S_{\lambda 2} = \frac{S_\omega - S_{\varphi 1} - S_{\varphi 2} - \sqrt{(S_\omega - S_{\varphi 1} - S_{\varphi 2})^2 - 4\omega}}{2} \tag{3.9b}$$

where the discriminant cannot be negative to avoid oscillatory behaviours,

$$(S_\omega - S_{\varphi 1} - S_{\varphi 2})^2 \geq 4\omega \tag{3.10}$$

The conditions identified in [BeEn06] serve as a general guideline when implementing an inertia weight, while the range of values is very different when including the constriction coefficient as the latter affects the weights as well [KeEb01]. In practice many different combinations are commonly used with the original values ($S_{\varphi 1} = S_{\varphi 2} = 2$) and those proposed by Clerc ($S_\omega = 0.729$ and $S_{\varphi 1} = S_{\varphi 2} = 1.49445$) being the most popular [SeMa09].

3.2.4 Neighbourhood Topologies

The performance of the PSO algorithm depends on the interconnections between particles [Kenn99a] [Kenn07]. The connections are generally known as the *neighbourhood topology* of the algorithm and form the basis for particle-to-particle communication that guides the search processes in PSO. There are two major classes of topologies known as the global topology, *global best* (gBest), where all the particles are interconnected (see Fig. 3.5), and the local topology, *local best* (ℓBest), where particles are only connected to a subset of the swarm known as the particle's neighbourhood. The size of the neighbourhood is denoted by S_n and refers to the $S_n - 1$ connections plus the particle itself [Mend04]. In order to analyze these various topological configurations, it is often convenient to describe them in terms of connected, simple, unweighted, undirected graphs where each vertex corresponds to a particle and each edge represents a connection between two particles in the same neighbourhood [Mend04]. In this form, one can describe the various configurations and utilize knowledge from other fields (i.e., networks [WaSt99]) to gain insight into the behaviour of the swarm.

3.2.4.1 Global Topology (gBest)

A gBest topology is often represented as a fully connected graph as shown in Fig. 3.5. At each iteration of the algorithm, S_g in Line 6 of Algorithm 3.2 is calculated and used to calculate the new position of the particles in the set using Eq. 3.2. In this case, S_g contains the best position in the entire set, thus accelerating convergence as all particles agree on where the current best solution is found. However, in doing so, the swarm risks premature convergence because the particles do not get to explore as much of the parameter space.

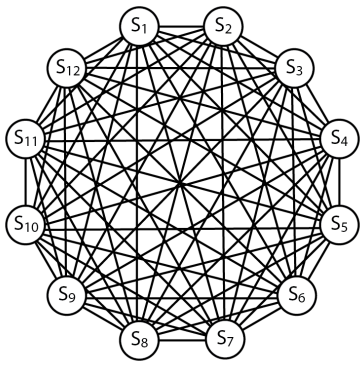


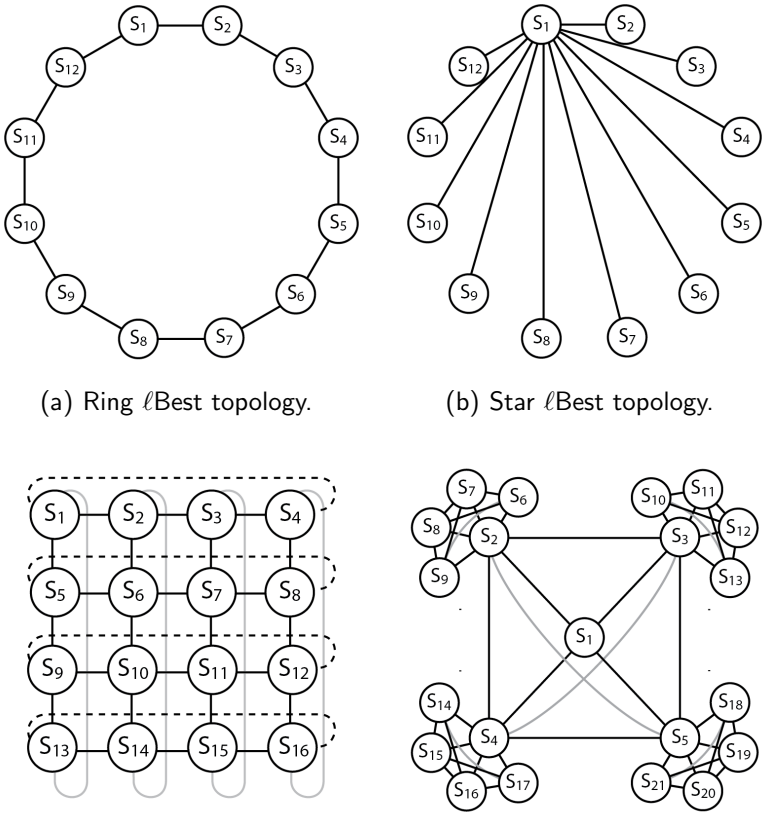
Fig. 3.5: Graphical representation of gBest topology in PSO.

3.2.4.2 Local Topologies (ℓBest)

In contrast, ℓBest topologies can be represented in many different ways that are either static for the duration of the algorithm, or that change over time based on the positions of the particles in the set [KeEb01]. For example, a dynamic ℓBest configuration can defined a neighbourhood of particles that are located within a certain distance from each other as defined by Eq. 3.11. In this case, the second norm measurement indicates the distance or the relative potential energy of the particles on each other. Although this topology appears to follow natural biologically-inspired behaviours, it is rarely implemented in practice because it requires that the distance from each particle to all other particles in the set be calculated [KeEb01]. To simplify the problem, ℓBest topologies are often fixed for particular algorithms with the most common configurations being ring, stars, torus, and hierarchical-clusters [Kenn99a] [KeMe02] [ScKi10].

$$distance < || \mathbf{S}_{x,k}(n) || \tag{3.11}$$

In the ring arrangement, the S_K particles are arranged in a ring-like formation with each particle, S_k, communicating with the adjacent $\lfloor \frac{S_n - 1}{2} \rfloor$ neighbours on either side as



(a) Ring ℓBest topology.

(b) Star ℓBest topology.

(c) Torus ℓBest topology.

(d) Hierarchical-cluster ℓBest topology.

Fig. 3.6: Graphical representation of ring, star, torus, and hierarchical ℓBest topologies in PSO. (a) Ring representation where each particle is connected to the adjacent neighbours. (b) Star where a root particle connects to all others, while individual particles only connect to the root. (c) Torus representation shown as a grid with wrap-around connections. (d) Hierarchical-cluster representation showing clusters connected to a root node.

shown in Fig. 3.6(a). For example, if $S_K = 30$ and $S_n = 5$, the neighbours of particle $S_k = 29$ are particles $S'_k = \{27, 28, 30, 1\}$. The advantage of this system is that each set of neighbours explores the parameter space based on different neighbourhood criteria. But, as a consequence, the convergence is much slower because information about good solutions can take many iterations to propagate through the network [KeEb01]. The star configuration selects one particle as the root that contains global knowledge and uses that

particle's information to update the rest of the set as depicted by Fig. 3.6(b) [KeEb01]. The "root" can act as a filter of information for the remaining particles and is often selected at random at the beginning of each run to avoid biasing results [Kenn99a]. Finally, the torus appears like a grid configuration with edges wrapped around to form a doughnut-like mesh as displayed in Fig. 3.6(c). This structure allows for local knowledge to be shared, while the overall global information can be propagated to the entire system within only a few iterations [KeMe02]. The hierarchical-clusters topology is a modified tree that creates a combination of small clusters that can explore the parameter space and then share information globally through the connected network as shown in Fig. 3.6(d) [ScKi10]. At the extremes, the root node only communicates with its children and any leaf only talks to its siblings. This topology draws on the benefits of the star topology where all the information is filtered through a root node, but at the same time benefits from small clusters that share information. Thus, the number of iterations until a good solution is propagated through the entire tree can be approximated to $\lceil 2\log_{S_n}(S_K) - 1 \rceil$ (obtained from traversing the tree from a leaf node to the root and back to another leaf node) [ScKi10]. Note that in star and hierarchical configurations, S_n is different for the root and children nodes.

In addition to the many variations of fixed ℓBest topologies, there are also various studies in which random topologies are used. The random topologies use a substrate connected graph (like a ring) and then add or switch a small random set of edges [Kenn99a], [KeMe02]. These types of graphs follow the *small-world* (SW) graphs that can delay information about good solutions for a few iterations in order to allow for exploration of the parameter space by other particles, while also providing multiple paths through the clusters for information flow [WaSt99]. This structure also creates small clusters that share a great deal of information. The one restriction that is key in generating these graphs is that they must remain connected; otherwise not all the particles can converge to the same point [Kenn99a].

3.2.4.3 Understanding the Impact of Topologies

The merits of each topology can be evaluated in terms of (i) the computational needs, (ii) the behaviour of the swarm, and (iii) the impact on the optimization algorithm. As with most such problems, there are tradeoffs between these alternatives that are worth considering to determine what topology to implement.

The computational complexity of gBest is described in Sec. 3.1.3 with some proposed improvements that exploit the topology and precompute both S_g and the fitness of each particle. When it comes to local topologies, the problem increases in that every particle has a unique S_g based on the particular neighbourhood. Therefore, the number of neighbours, S_n, affects the calculations that are approximated as $\mathcal{O}(N_{max}S_K S_n D^2)$, where $S_n < S_K$. It is possible to design a dynamic programming solution for each local topology, but the amount of memory required to cache all the parameters increases significantly.

The behaviour of the swarm was studied by Kennedy based on the work of Watts and Strogatz to determine the interactions between agents in a network-like environment [WaSt99] [Kenn99a]. That study finds that the *length* and *clustering coefficient* are two measures of a network configuration that give some insight into the flow of information for a given topology [WaSt99]. The length measures the minimum number of edges that connect any two particles to each other, while the clustering coefficient is an indicator of the degree of cliquishness and is defined as the ratio of the number of edges connected to a given particle over the total number of links, assuming a gBest topology.

To test the flow of information based on the length indicator, Mendes and Kennedy [MeKN04] generated random graphs with different lengths and also compared many of the ℓBest topologies described in Sec. 3.2.4.2. They found that the flow of information in the torus works best as the information takes a few iterations to spread throughout the network while also providing a high degree of connections and links with multiple paths that enable

particles to escape local solutions. This was verified and compared to the hierarchical-clusters topology to demonstrate the trajectories of particles as they explore more of the parameter space while good solutions get propagated through the network [ScKi10].

Finally, in terms of the optimization, Kennedy and Eberhart [KeEb01] propose the general rule that local knowledge leads to greater exploration of the parameter space with slower convergence due to particles being influenced by different neighbourhoods. Global knowledge, on the other hand, can lead to local optimum because all the particles follow the best solution at each time step.

3.2.5 Stopping Criterion

In all unconstrained non-linear problems, it is hard to determine when the algorithm should stop, and be either satisfied with the solution found or give up. The most popular techniques found in literature regarding PSO set an upper limit for the number of iterations and compare the results to a known optimal solution. The first technique mentioned is very useful in real-time applications where it is generally desired to find a solution within a particular amount of time [ZiPL05]. The notion of comparing the results against a known solution is adequate for benchmarking implementations of an algorithm, but this technique is not applicable to real problems where a solution is not known [ZiPL05].

If the optimum is not known, one way to determine when to stop is to study the changes in the evaluated fitness function over time. If the result do not change by more than a given convergence error, ϵ, over a pre-defined number of iterations, then the algorithm has converged to a value [ZiPL05]. Other alternatives include measuring the distribution of the particles in the parameter space or terminating when the step size for all particles is smaller than a predetermined value [ZiPL05].

3.2.6 Other Variations

In addition to the aforementioned variations presented in this section, there are many other variations that aim to improve PSO for specific applications. Some of these include using chaos in the parameter exploration to prevent repeating the path taken by particles similar to the improvements to SA [ShKi96] [LiuW05]. In an effort to prevent premature convergence, the *Attractive Repulsive PSO* alternates between attractive and repulsive phases of different durations by subtracting terms in Eq. 3.2, thus being able to escape some local solutions [RiVe02]. The *Binary PSO* provides a discrete alternative to the continuous optimization method [KeEb97]. More options are described in the survey paper by Sedighizadeh and Masehian [SeMa09].

3.3 Summary

This section provided a thorough review of the existing PSO algorithm, its variations, and behaviour. In Ch. 4, the understanding of the behaviour of PSO is further refined through a novel trajectory analysis that allows us to compare the effect of different parameters, implementations, and variations beyond simply using the number of iterations until the algorithm converges. Furthermore, these concepts are used to select parameters for a real-time scheduling algorithm based on PSO.

Chapter 4

Scheduler System Design

The design of a *symmetric multiprocessing* (SMP) real-time scheduler based on *evolutionary algorithms* (EA) is divided into two main parts: (i) the EA and (ii) the scheduler. In the context of the optimization algorithm, one must first define some general constraints that help select an algorithm as well as the specific parameters to be used. Then, the algorithm is selected, implemented, and verified using various parameters to suit the needs of the problem, as described in Sec. 4.1. Finally, to incorporate this into a scheduler, one needs to define a means of encoding the scheduling problem into the context of the algorithm and devise a cost function to guide the search for an optimal solution, as discussed in Sec. 4.2.

4.1 Optimization Algorithm

4.1.1 Requirements for Optimization Algorithms in Real-Time Systems

There are many trade-offs involved in the design of an algorithm that relate to speed, memory, and other parameters. Identifying some of the high level desirable features for

an optimization algorithm used for scheduling tasks in real-time systems is of paramount importance to help select the algorithm, adjust parameters to desired behaviours, and ultimately obtain results that satisfy the real-time constraints of the problem. One can identify five major areas to study: (i) convergence rate, Q_α, (ii) degree of exploration, Q_β, (iii) storage and system size, Q_γ, (iv) adaptability, Q_δ, and (v) multi-scale features, Q_ϵ [Kins04] [Kins05b]. Note that the goal is to identify the requirements for real-time scheduling in space systems. The following sections describe some elements of each of these five key areas of interest.

4.1.1.1 Convergence Rate

A fast *convergence rate*, Q_α, is the most common requirement and means of comparing optimization algorithms. It is expressed as the number of iterations until a termination criterion is reached [ShKi96] [KeEb01] [Boyd04] [FlMa08]. This measure implicitly requires that all iterations last approximately the same amount of time such that the number of iterations can be compared using a linear scale. The convergence rate is negligible in static schedules that can be computed *a priori*, while any system that must adjust the schedule at run-time must do so quickly to avoid missing deadlines.

Many authors use Q_α as the primary metric for comparing different evolutionary optimization algorithms [KeEb01] [FlMa08]. Unfortunately, this does not provide enough information to judge an algorithm effectively over a large set of problems because EAs depend on interactions, feedback, and stochastic processes that vary drastically for different approaches [ScKi11a]. Furthermore, the algorithm convergence might not yield an acceptable solution, and, therefore convergence rate along should not be used for comparison purposes. Instead, one must use a variety of metrics (including those described in this thesis) to get a more complete picture. The Q_α is sufficient when testing small variations

to a subset of parameters in the algorithm that do not change the overall nature of the algorithm [ScKi11a]. For example, testing the same GAs with different mutation rates can be efficiently compared as the effect does not vary the overall performance of the algorithm. Furthermore, even when comparing single parameter changes one must consider trade-offs that affect other elements of the implementation. An example of that is the increased memory required when using more particles in PSO to speed up convergence.

4.1.1.2 Degree of Exploration

The *degree of exploration*, Q_β, gauges how much of the parameter space is explored by an optimization algorithm in the search for a solution. The degree of exploration is useful for comparing algorithms that take approximately the same number of iterations to converge [ScKi11a]. An argument could be made that if two algorithms have the same Q_α, then they both explore the same number of solutions, and thus, the Q_β of the parameter space is the same. The problem with this argument is that it assumes that the parameter space is uniformly distributed such that the optimal solution is equally likely to be found at any point. However, from looking at sample parameter spaces like those described in Sec. 4.1.3, it is possible to have valleys of local solutions where the algorithms can be trapped (i.e., see Fig. 3.4).

The population-based EAs are inherently more likely to explore more of the parameter space if the initial solutions are randomly distributed [Ange98]. For example, as highlighted in Sec. 3.2.4, local topologies in PSO can lead to higher degrees of exploration because it takes time for all the neighbourhoods to learn about a good solution found by an individual particle in the swarm [ScKA10]. In the case of memetic algorithms, the degree of exploration is very high during the global search portion and then decreases for a local search to refine the solution [FlMa08]. In contrast, SA explores a single solution at a time; therefore,

issues of repeated paths are of high concern leading to significant improvements like chaotic trajectories [ShKi96].

In real-time systems, EAs are often implemented with a maximum number of iterations, N_{max}, to ensure that the worst-case execution time is well defined for the purpose of scheduling tasks. Therefore, it is better to implement algorithms that explore more of the parameter space in the same amount of time in order to improve the capabilities of the search within the same number of iterations.

Despite its importance, the literature review did not identify metrics capable of quantifying the degree of exploration achieved by an EA because every function optimized would pose different constraints on the problem [WoMa97]. Therefore, new techniques are required to move towards better representations of the behaviour of EAs that would lead towards appropriate choices for algorithms and their respective parameters that would meet the exploration requirements of the problem at hand [ScKi11a].

4.1.1.3 Storage and System Size

The *amount of resources* used refers to the storage and system size, Q_γ, used by an optimization algorithm during the search process [ScKi11a]. The resources consist of a combination of both memory used to store the state of the algorithm at each iteration and the computational complexity of the algorithm. These two ideas are inherently linked such that one is often making trade offs between improving the performance by caching more information through dynamic programming techniques or recomputing many values to save space. Either way, it is a critical area that must be assessed for a particular application based on the parameters of the problem at hand.

In the case of memory storage, the main concerns lies in the realm of population-based EAs, where many solutions are stored with their associated parameters. For example,

solving a 2-dimensional problem in PSO with $S_K = 20$ using floating point arithmetic requires approximately 480 bytes just to store the particles (estimated as 20 *particles* $\times$ 2 *dimensions* $\times$ [1 *position* + 1 *velocity* + 1 *memory*] $\times$ 4 *bytes*). This grows to 720 bytes for $S_K = 30$ highlighting the trade off between memory and exploring more of the parameter space. This difference is more pronounced for high-dimensional problems. In real-time systems, this amount of *random access memory* (RAM) for a problem is very significant and can take away resources from other tasks on the system. This is worse if one tries to optimize the computational performance of the algorithm by storing more information as described in Sec. 3.1.3.

Each application dictates how the trade off should be performed and which of the two components should be given a higher priority. For real-time systems, one wants to minimize both parameters and thus must examine the specifics of the algorithm to select the best option for the particular application.

4.1.1.4 Adaptability

The adaptability, Q_δ, of an optimization algorithm refers to the ability to adapt to the variations in the parameter space to ensure good solutions are found for most scenarios [ScKi11a]. This problem is further described in terms of: (i) the large problem spaces with many candidate solutions to evaluate, (ii) asymmetrical high-dimensional problems where it is difficult to determine which variables, and which combinations of variables, to modify, (iii) complex nonlinear fitness functions with many local optima and/or discontinuities, and (iv) the changes experienced by the parameter space over time (even when assuming the changes do not affect a single optimization run) [EbSh07]. Consequently, one must find a tangible way to optimize functions in real-time systems when not all the information is available in advance and there are unknowns about the details of the parameter space.

Intuitively, one can use a large number of test functions, like those selected in Sec. 4.1.3, to ensure the algorithm can perform under different scenarios. But, doing so leads to another problem described by Wolpert and Macready as the *no-free-lunch* (NFL) theorem [WoMa97]. The NFL theorem states that even if one shows than an algorithm works well on one class of problems, another class of problems that was not fully considered offsets its performance for the algorithm in question. This means that one cannot define an overall general optimizer that works on any application. Therefore, rather than solving general cases, it is important to incorporate problem-specific (parameter space) knowledge into the behaviour of the algorithm to improve its performance (convergence and speed) [WoMa97].

This thesis utilizes a dual approach that first tests the implementation of the algorithm on four well-known functions to verify and validate the implementation as described in Sec. 4.1.3 – 4.1.4. Then, specific parameters are selected for the scheduling problem being solved as described in Sec. 4.2. However, this is limited to test cases for the tasks being scheduled, and thus sets with different characteristics (i.e., different number of tasks or distribution of tasks) may require additional off-line testing to adjust parameters, like S_{Vmax}, so that the particles can escape local solutions for the given parameter space.

4.1.1.5 Multi-scale Capabilities

One desirable characteristic for all EAs is to show that the solutions being generated are actually evolving and improving over time. Although it appears obvious, this implies that one needs smart perturbations for candidate solutions to generate solutions that might improve the current best solution found. This type of behaviour is apparent through multi-scale analysis, Q_ϵ, that extracts long-term correlations in the trajectories of candidate solutions [ScKi11a]. In-depth reviews of multi-scale metrics are provided in [Kins04] [Kins05].

Algorithms displaying these characteristics can be exploited to improve the performance of the algorithm by means of predicting future better states and fast-tracking the algorithm as was done in cellular automata populations in [GrKi98].

In the context of real-time scheduling presented in this thesis, long-term correlations in the trajectories are clear indicators that the candidate solutions are not random but are rather approaching a valid schedule that meets all the real-time requirements of the system. This is specially important in SMP scheduling as the number of possible schedules makes the problem intractable.

4.1.2 Selection of Evolutionary Algorithm

There are many algorithms that meet some of the constraints outlined in Sec. 4.1.1. The reason for selecting PSO is that the perturbations are generated primarily from information collected by the swarm in previous iterations multiplied by small stochastic weights to provide some randomness to explore the parameter space. Such behaviours are not available in other EAs considered. For example, the SA perturbations are accepted based on the Boltzman criterion, while the GAs use crossovers and a selection process to accept candidate solutions. The reason for selecting an algorithm that guides the perturbations is because this can lead to further improving the convergence rate by identifying long-term correlations to skip through many iterations.

The analysis presented in Sec. 4.1.4 shows that the perturbations that generate the step size exhibit directed movements towards a solution. Rudimentary experiments of that nature for SA and GAs do not exhibit the same properties and, instead, show small or no correlation between solutions. This empirical study indicates that there may be some advantages in terms of the adaptability and multi-scale capabilities for PSO that make it suitable for real-time systems.

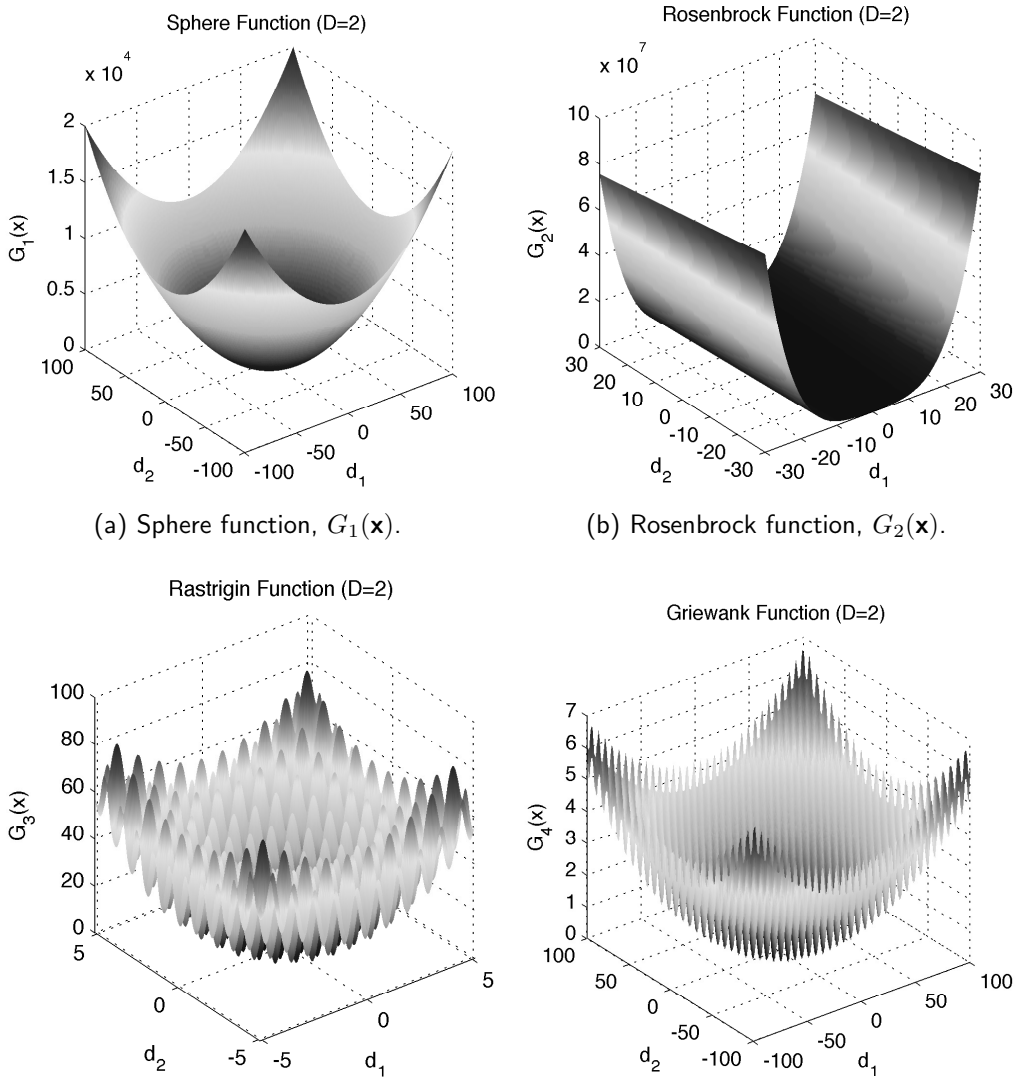
Other criteria are not addressed in this thesis explicitly (e.g., convergence rate and system size), as those constraints are addressed by other authors [KeEb01] [BrKe07]. These constraints are very much coupled to the parameter space being optimized, and therefore more complex environments may require more resources to find optimal solutions [ScKi11a]. The degree of exploration is shown in the time series analysis in Fig. 4.10, and is described in more detail in [ScKA10]. This study compared different parameters for the algorithm and demonstrated how some conditions can take the same number of iterations to converge while exploring more of the parameter space through the use of local topologies with small clusters of particles (i.e., hierarchical-clusters topology).

4.1.3 Selection of Test Functions

The PSO algorithm is verified by comparing the results to those available in the literature for well known problems in non-linear, unconstrained optimization problems. The functions selected are the *Sphere* ($G_1(\mathbf{x})$), Rosenbrock ($G_2(\mathbf{x})$), Rastrigin ($G_3(\mathbf{x})$), and Griewank ($G_4(\mathbf{x})$) as shown in Fig. 4.1 and used in [Jong75] [ShEb99] [KeEb01] [Trel03] [BeEn06]. These functions have special properties as described below that make them suitable for validating and experimenting with the PSO algorithm.

The Sphere function is given by Eq. 4.1. It is a perfect convex function where all points are monotonically decreasing towards the optimum value as shown in Fig. 4.1(a) for $D = 2$. The results from optimizing this function serve as a benchmark for more complicated examples. The global minimum is $G_1(\mathbf{x}) = 0$ and is located at $\mathbf{x} = \{0, 0, \ldots, 0\}$. The initial domain for the function is commonly set to $[-100, 100]^D$ along every axis (i.e., $S_{Xmax} = 100$).

$$G_1(\mathbf{x}) = \sum_{d=1}^{D} x_d^2 \tag{4.1}$$



(a) Sphere function, $G_1(\mathbf{x})$.

(b) Rosenbrock function, $G_2(\mathbf{x})$.

(c) Rastrigin function, $G_3(\mathbf{x})$.

(d) Griewank function, $G_3(\mathbf{x})$.

Fig. 4.1: Test functions selected to verify the implementation of PSO. (a) Sphere function defined by Eq. 4.1. (b) Rosenbrock function defined by Eq. 4.2. (c) Rastrigin function defined by Eq. 4.3. (d) Griewank function defined by Eq. 4.4. Note the Griewank function is only plotted for a fraction of the defined domain in order to visualize the local peaks. The overall envelope for the function continues in the same convex manner.

The Rosenbrock function is given by Eq. 4.2. It is also a unimodal function that features steep edges along one dimension and very gentle slopes along the other dimension as shown in Fig. 4.1(b). This leads to one dimension being optimized very quickly, while the other can take time and sometimes trap EA implementations that use improvements in fitness as a terminating condition. The global minimum is $G_2(\mathbf{x}) = 0$ and is located at $\mathbf{x} = \{1, 1, \ldots, 1\}$. The initial domain for the function is commonly set to $[-30, 30]^D$ along every axis.

$$G_2(\mathbf{x}) = \sum_{d=1}^{D} \left(100 \left(x_{d+1} - x_d^2\right)^2 + \left(x_d - 1\right)^2\right) \tag{4.2}$$

The Rastrigin function is given by Eq. 4.3. Figure 4.1(c) shows an envelop function that resembles a gentle convex function that guides the global search through the very rough local searches. This function has many local solutions that can trap an algorithm. The optimum value is $G_3(\mathbf{x}) = -180$ and is located at the origin. The initial domain for the function is commonly set to $[-5.12, 5.12]^D$ along every axis. The smaller domain implies that small step sizes are required to get out of the local solutions.

$$G_3(\mathbf{x}) = \sum_{d=1}^{D} \left(x_d^2 - 10\cos\left(2\pi x_d\right) + 10\right) \tag{4.3}$$

Equation 4.4 for the Griewank function is the most complicated function in the set. It has similar features to the Rastrigin function in a general convex shape with many local peaks with relatively smaller variations in the cost compared to the Rastrigin function make it harder for algorithms to detect changes as they converge on a solution. The optimum value is $G_4(\mathbf{x}) = 0$ and is located at the origin. The initial domain for the function is commonly set to $[-600, 600]^D$ along every axis.

$$G_4(\mathbf{x}) = \frac{1}{4000} \sum_{d=1}^{D} \left(x_d^2\right) - \prod_{d=1}^{D} \left(\cos\left(\frac{x_d}{\sqrt{d}}\right)\right) + 1 \tag{4.4}$$

Symmetric domains were selected for all functions in order to compare the results to those from the literature for similar implementations. However, as noted in [FoBe95], this can bias the results towards faster convergence rates than asymmetric domains.

There are many other functions that could have been considered including Ackey, Schaffer's f6, and others that feature very flat surfaces except around the solution and, therefore, could also prove valuable. Although adding more functions would show the resilience of the algorithm to certain types of features, it would not guarantee a general purpose optimization algorithm as per the NFL theorem. For the purpose of validating the implementation, the set of test functions provided confirmation through many publications and included many features that would be applicable in testing a cost function for a real-time scheduler as described in Sec. 4.2.3.

4.1.4 Analysis of Particle Trajectories

As mentioned in Sec. 4.1.1.1, evaluating EAs through the convergence rate alone is insufficient as there are many other desirable features that can give insight into the performance of the algorithm when exposed to other more complicated cost functions. Therefore, one needs to study the behaviour of the algorithm through the trajectory of the candidate solutions evaluated to observe the patterns that emerge in the search for a global optimum. In population-based EAs like PSO, there are many candidate solutions at each iteration that are coupled together through the update equations. Therefore, as a tangible first step, one can study the trajectory of a single particle to learn about its characteristics as they relate to the behaviour of the entire swarm.

4.1.4.1 Selecting a Representative Particle to Study

For a trajectory analysis to be meaningful, the particle's behaviour must be representative of the set. Figures 3.3 and 3.4 show the trajectory of a particle through the parameter space in the Sphere and Rastrigin functions respectively. The obvious question is then, *are these behaviours typical for the particles?* This is a rather difficult question to answer because of the stochastic nature of the algorithm embedded through the random initialization, varying weights ($S_{\varphi 1}$ and $S_{\varphi 2}$) multiplied by random numbers, and the effect of social interactions.

In [KeEb01], [Berg01] and [Trel03], the trajectories of particles were selected to show convergent and divergent behaviours under different parameters. To show this, each dimension can be plotted independently as the update equations only couple dimensions through the cost function [Trel03]. This is further exploited for symmetric (i.e., Sphere and Rastrigin) functions where a single dimension may provide insight into the problem. However, when working with asymmetrical problems like the Rosenbrock function, the trajectories along each dimension must be analyzed separately to obtain the full behaviour.

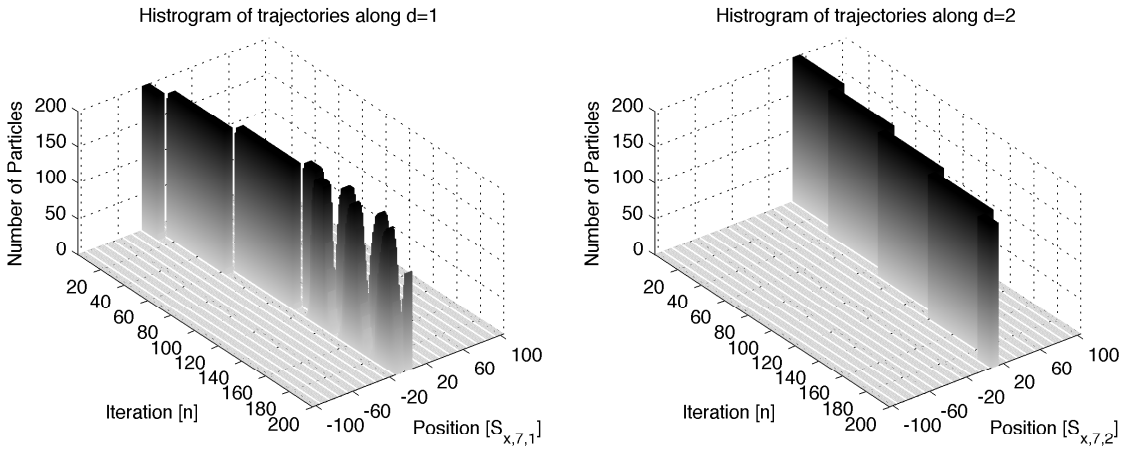
In terms of reducing the number of unknowns to establish typical trajectories, one can remove the stochastic parameters for $S_{\varphi 1}$ and $S_{\varphi 2}$. The simplest case consists of setting both the random number generators to the mean expected value of 0.5 so that there would be an even distribution of personal and social influences [Trel03]. This reduces Eq. 3.2 to a deterministic system, and although easy to process, it removes much of the exploration performed of individual particles. Learning about the deterministic behaviours, one can then re-introduce the random number generator to evaluate over different particles as shown in [Berg01]. This approach suggested in the literature does not address the question of whether the particle studied is representative of the swarm.

4.1.4.2 Extended Time Series Analysis

This thesis proposes a novel method to select candidate trajectories to study consisting of fixing the initial conditions for PSO (i.e., the position and velocity) and running the algorithm N_{runs} times to produce an ensemble. This is accomplished by seeding the random number generator, initializing the particles and storing the results. Then, re-seeding the random number generator and running the algorithm with stochastic weights. The objective is that although the amount of exploration by individual particles might vary in each iteration, the emergent behaviour of the swarm should persist through the different runs [Forr91].

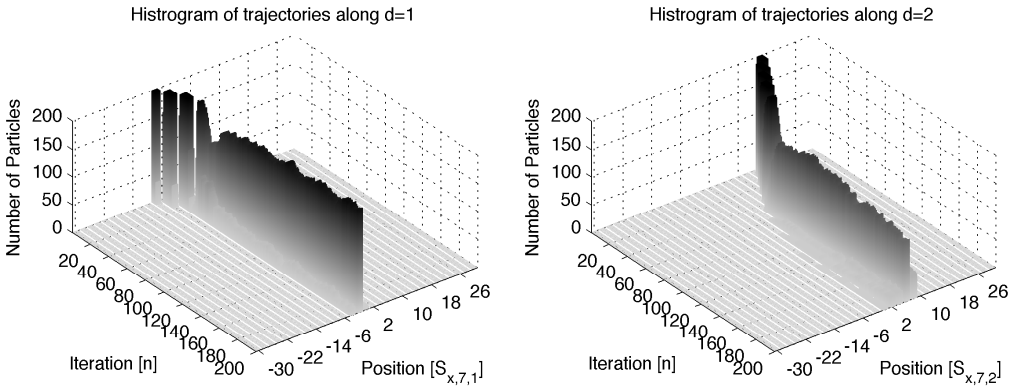
Before extracting individual trajectories, it is important to understand how the same initialization performs for different runs of the algorithm. Figures 4.2-4.5 show histograms over time for the set of test functions along each dimension. These images are generated by performing $N_{runs} = 200$ runs where the particles are initiated to the same position and velocity vectors before changing the seed for the random number generator that drives the social and personal weights. For these runs, the algorithm was purposely slowed down using $S_{Vmax} = 0.2$ as proposed by [KeEb01] as a means of prolonging the trajectories available to study. Note that all the runs on all four functions returned the global optimal solution with varying decimals of precision that matched the expected values from [KeEb01], [ShEb99], [Ange98]. The figures confirm that fixing the initial conditions, the random weights can allow for some individual behaviours, but the overall behaviour of the particle is primarily dictated by the interactions with the neighbours. Note that for these diagrams, $S_{x,7}$ was used as all particles in the set showed similar behaviours when taking into account the different conditions after initialization.

More specifically, the histograms confirm the expected behaviours described in Sec. 4.1.3 for particles travelling along each function. The behaviour in the Sphere function shows



(a) Position $S_{x,7,1}$ evaluated for $G_1(\mathbf{x})$. (b) Position $S_{x,7,2}$ evaluated for $G_1(\mathbf{x})$.

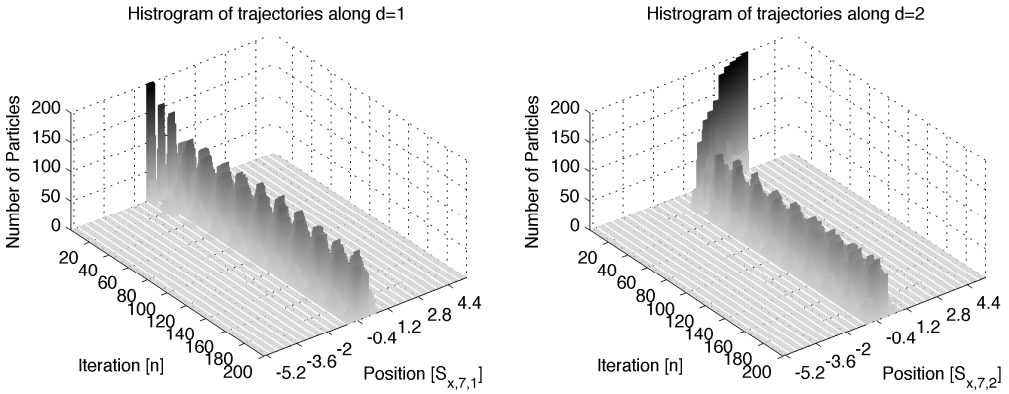
Fig. 4.2: Histogram showing the trajectory of an ensemble of $N_{runs} = 200$ particles initialized to the same conditions in the Sphere function. (a) Position along $d = 1$ dimension, $S_{x,7,1}$. (b) Position along $d = 2$ dimension, $S_{x,7,2}$.



(a) Position $S_{x,7,1}$ evaluated for $G_2(\mathbf{x})$. (b) Position $S_{x,7,2}$ evaluated for $G_2(\mathbf{x})$.

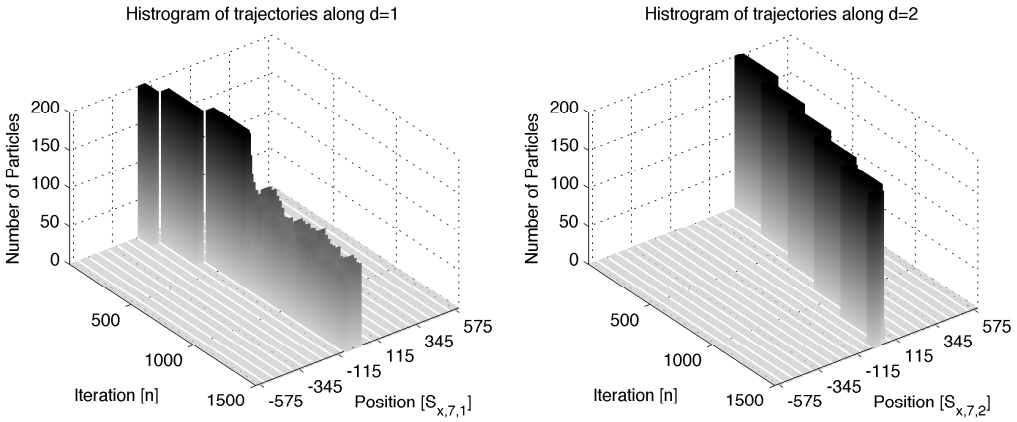
Fig. 4.3: Histogram showing the trajectory of an ensemble of $N_{runs} = 200$ particles initialized to the same conditions in the Rosenbrock function. (a) Position along $d = 1$ dimension, $S_{x,7,1}$. (b) Position along $d = 2$ dimension, $S_{x,7,2}$.

the $d = 2$ axis being optimized without many oscillations in Fig. 4.2(b), while the $d = 1$ oscillates towards the solution as shown in Fig. 4.2(a). Figure 4.3 supports the notion that the asymmetric Rosenbrock function experiences very rapid convergence with small



(a) Position $S_{x,7,1}$ evaluated for $G_3(\mathbf{x})$. (b) Position $S_{x,7,2}$ evaluated for $G_3(\mathbf{x})$.

Fig. 4.4: Histogram showing the trajectory of an ensemble of $N_{runs} = 200$ particles initialized to the same conditions in the Rastrigin function. (a) Position along $d = 1$ dimension, $S_{x,7,1}$. (b) Position along $d = 2$ dimension, $S_{x,7,2}$.



(a) Position $S_{x,7,1}$ evaluated for $G_4(\mathbf{x})$. (b) Position $S_{x,7,2}$ evaluated for $G_4(\mathbf{x})$.

Fig. 4.5: Histogram showing the trajectory of an ensemble of $N_{runs} = 200$ particles initialized to the same conditions in the Griewank function. (a) Position along $d = 1$ dimension, $S_{x,7,1}$. (b) Position along $d = 2$ dimension, $S_{x,7,2}$.

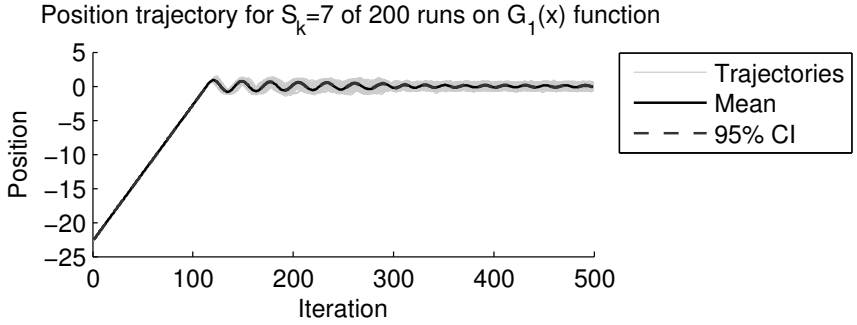
variance along $d = 1$ compared to $d = 2$ where changes are less noticeable because of the gentle slope in the parameter space. The multimodal functions show a lot more variations in the histogram of the Rastrigin function, Fig. 4.4, compared to the Griewank function,

Fig. 4.5 because the local solutions have a greater area for the particle to get stuck in.

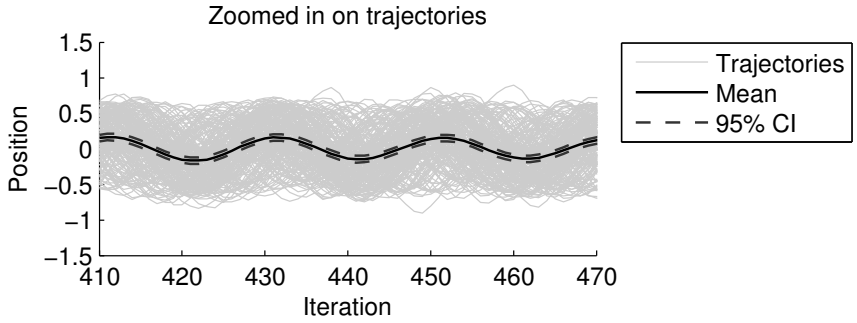
The global patterns that emerge in each histogram confirm that fixing the initial conditions can help select candidate particles to perform a *data-driven trajectory study to understand the behaviour of PSO*. Given the histogram, a particle is selected from the set of $N_{runs} = 200$ runs to use in the trajectory analysis. As evident from the histograms, there is still some variation in the trajectories, so it is important to select a particle within the set of runs that exhibits the typical behaviour representative of the set. To do this, the trajectories along each dimension are plotted on a graph along with the mean, $\mu_{x,k,d}(n)$ with its 95% confidence interval for each iteration. The results are shown in Figures 4.6-4.9 with additional images for other functions and zoomed in versions presented in Appendix A.

The trajectory in Fig. 4.6(a) shows a transient phase that has direct movement towards the solution, followed by a steady-state phase that attempts to perform a local search [ScKi11a]. Similar behaviours are shown for the Rastrigin function in Fig. 4.7(a). The mean and confidence intervals are calculated on the position along one dimension in $N_{runs} = 200$ runs of the algorithm for each iteration. Note that direct regression methods for the trajectories cannot be implemented because of the heteroskedastic properties of the trajectories that show an increased variance as the iteration increases [Gree11]. Therefore, the confidence interval is only used as a general guideline and is computed under the assumption that the trajectories are normally distributed.

The typical particles selected from the 200 runs of the algorithm are those that have the smallest *mean square error* (MSE) using $\mu_{x,k,d}(n)$ as the prediction of a nominal trajectory. That is, the typical particle performs very close to the average trajectory and, therefore is representative of the full set. For comparison, those with the worst MSE are also extracted as an atypical trajectory for a misbehaving particle. Figure 4.8 show the typical and worst behaving particle for the Sphere function. In this case, run 91 produced the closest looking



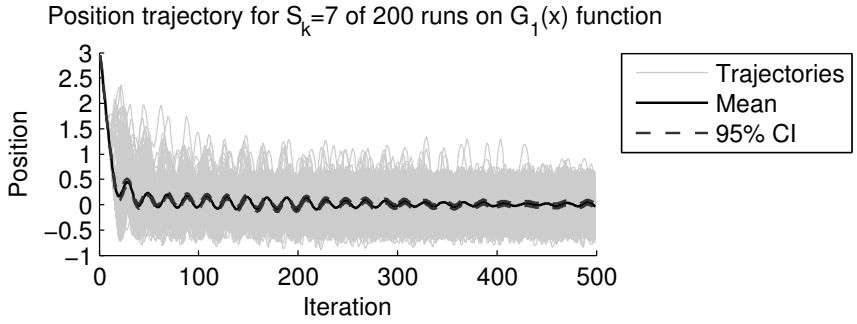
(a) Ensemble of $S_{x,7,1}$ for $G_1(\mathbf{x})$.



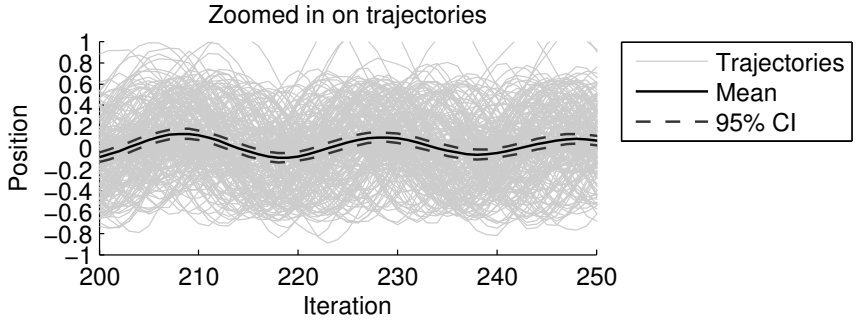
(b) Zoomed representation of Fig. 4.6(a)

Fig. 4.6: Identifying typical particle behaviours on Sphere function. (a) Position, $S_{x,7,1}$ plotted against fitness for 200 runs. The mean and 95% confidence interval are shown. (b) Zoomed in representation of Fig. 4.6(a).

trajectory to that of $\mu_{x,k,d}(n)$ so it is selected for study. As a general comparison, the worst behaving particle oscillated with a greater amplitude and appeared not to be synchronized with the other N_{runs} runs of the algorithm. This can be due to some early exploration as a result of transitioning to the steady state portion of the trajectory either faster or slower than the average particle. In contrast, the Rastrigin function shows 4 as the run that most resembled the average trajectory. It too remain very close to the average trajectory, however the worst run had a much greater variation. In this case, the variation of around 1 unit along $d = 2$ reveals that the particle is trapped in a local solution very close to the global optimum. Note that although this particle is stuck at a local solution, the algorithm



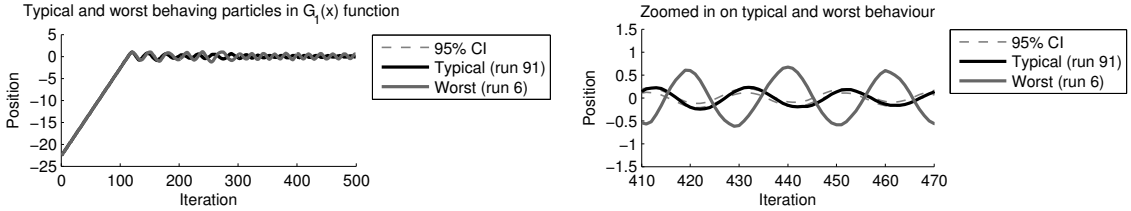
(a) Ensemble of $S_{x,7,2}$ for $G_3(\mathbf{x})$.



(b) Zoomed representation of Fig. 4.7(a)

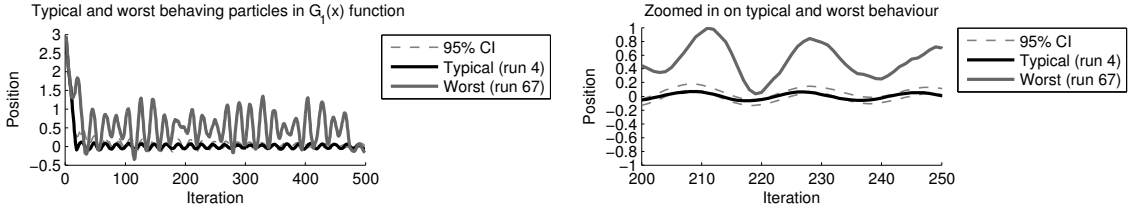
Fig. 4.7: Identifying typical particle behaviours on Rastrigin function. (a) Position, $S_{x,7,2}$ plotted against fitness for 200 runs. The mean and 95% confidence interval are shown. (b) Zoomed in representation of Fig. 4.7(a).

returned the global optimum because the majority of the particles had reach that point successfully. Similar results were obtained for the Rosenbrock and Griewank functions with the graphs shown in Appendix A. The only outstanding results obtained in the Rosenstock function were that the average particle was slowly approaching the global optimum along the gentle slope of $d = 2$, while the particle identified as the worst behaving particle reached the area surrounding the solution much faster for some of the particles studied. This was not consistent for all particles as it depended on their initial position on the parameter space.



(a) Typical and worst particles $S_{x,7,1}$ for $G_1(\mathbf{x})$. (b) Zoomed representation of Fig. 4.8(a)

Fig. 4.8: Identifying typical particle behaviours on Sphere function based on minimum MSE from the calculated trajectory mean, $\mu_{x,k,d}(n)$. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. 4.8(a).



(a) Typical and worst particles $S_{x,7,2}$ for $G_3(\mathbf{x})$. (b) Zoomed representation of Fig. A.17(c)

Fig. 4.9: Identifying typical particle behaviours on Rastrigin function based on minimum MSE from the calculated trajectory mean, $\mu_{x,k,d}(n)$. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. 4.9(a).

4.1.4.3 Extracting Particle Behaviours from Trajectories

Having selected a particle to study, the behaviour analysis consists of extracting the time series showing the position, $\mathbf{S}_x$, velocity (relative change in position), $\mathbf{S}_v$, and fitness over time. Figures 4.10 and 4.11 show the time series of interest for the particles traveling through the Sphere and Rastrigin function along the $d = 1$ and $d = 2$. In these figures, the top plot shows the particle position over time. The best positions found by the particles along each dimension are not shown but can be described as smooth curves that monotonically decrease towards the optimum point, while the position plots show the exploration of the particle over time. The middle plot shows the velocity of the particle over time. This plot highlights the bounds of the trajectory set to $S_{Vmax} = 0.2$ as a means of controlling the

convergence rate for the algorithm analysis. The last plot shows the fitness of the particle and the best position found by the particle over time. In the case of the Sphere function (shown in Fig. 4.10, the fitness shows little variation because of the simplicity of the $G_1(\mathbf{x})$ function being optimized. In contrast, more complex functions show how the fitness of the current position varies drastically as particles explore the parameter space.

The trajectories shown in Fig. 4.10 and 4.11 confirm the breakdown between the transient and steady state portion of the trajectories. The separation is important because the properties during the transient are completely different from those of the remaining of the time series [KaSc04]. The transient is defined as the time from the initial state until the time when the position of all the particles along all the dimensions reaches a threshold. In this case, the threshold is set to 300 iterations to match the point where the particle switches from a global to a local search. Note that the velocity slows down during certain portions of the trajectory (i.e., iterations 400-600 in Fig. 4.10) depending on the position of the global best. If the global best is very close, then the social influences are very small causing the velocity to decrease.

4.1.5 Single Particle Analysis

4.1.5.1 Time Domain Analysis

The first analysis performed on the time domain is to determine whether the time series extracted is stationary. If a stochastic process has constant first moment (mean) and second moment (variance or covariance, if necessary), then the process is called *wide-sense stationary* (WSS) or weakly-stationary. And, if all the moments of a stochastic process are unchanging, then the process is called *strict-sense stationary* (SSS) or strongly-stationary [KaSc04]. In PSO it is hard to determine whether the signal is SSS from a single time series as that requires an analytical solution. Instead, one can determine whether the signal is

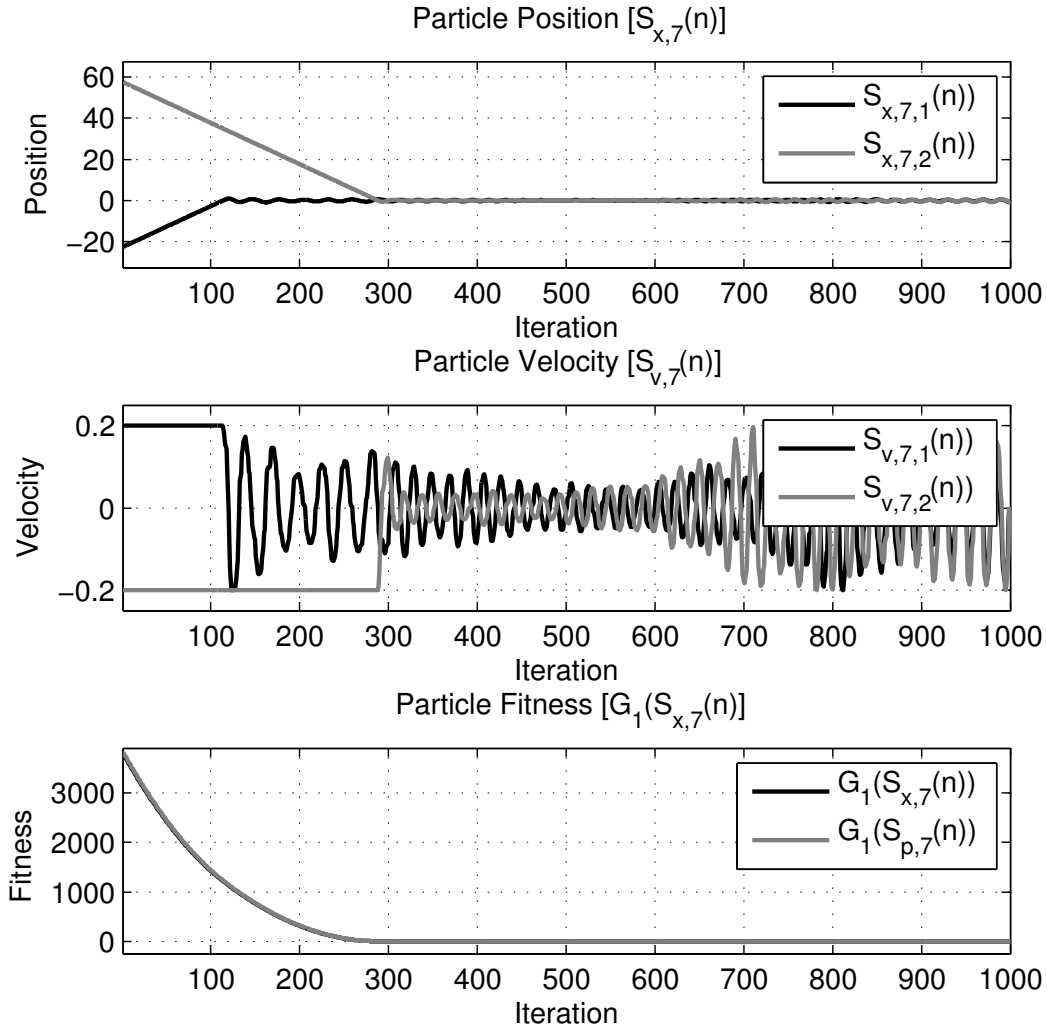


Fig. 4.10: Time series showing the particles position, velocity, and fitness over time in the Sphere function. (a) The particles position over time along the 2 dimensions being optimized. (b) The velocity of the particle over time along the 2 dimensions being optimized. (c) The fitness of the particle as it converges to the global optimum when the fitness equals zero. This plot highlights the monotonically decreasing fitness of the best position and contrasts the variation in the current positions fitness as the particle explores many of the local solution spaces.

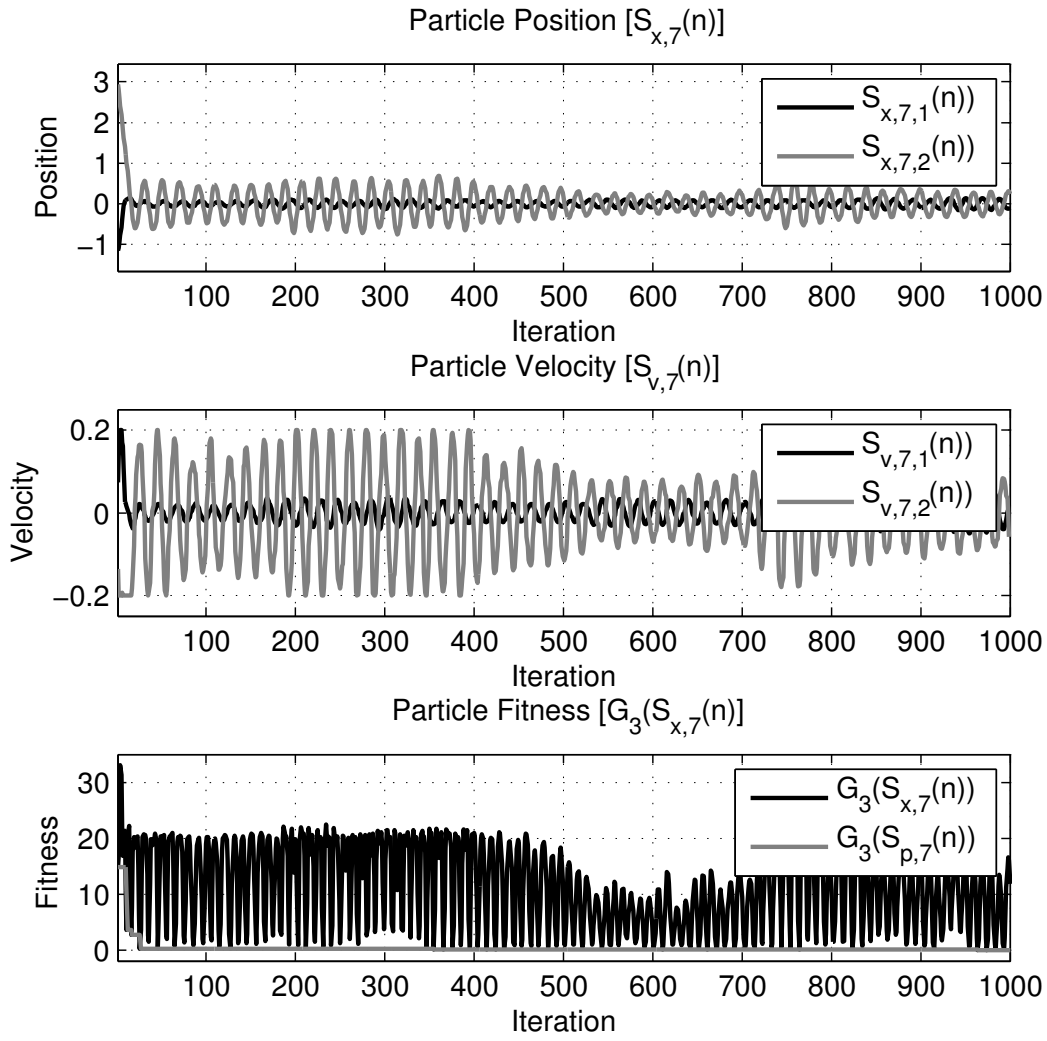


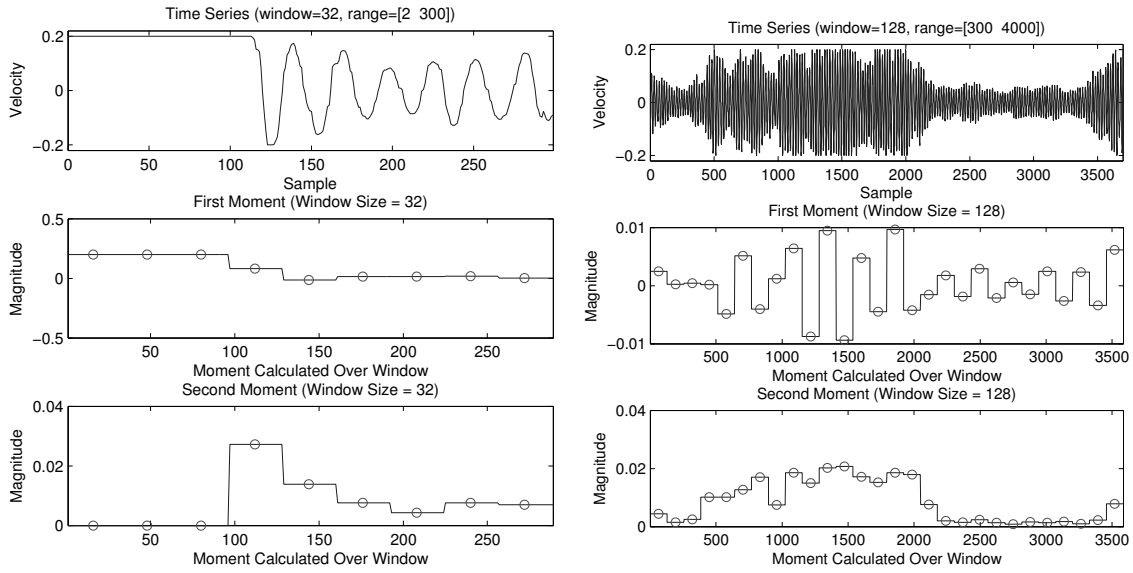
Fig. 4.11: Time series showing the particles position, velocity, and fitness over time in the Rastrigin function. (a) The particles position over time along the 2 dimensions being optimized. (b) The velocity of the particle over time along the 2 dimensions being optimized. (c) The fitness of the particle as it converges to the global optimum when the fitness equals zero. This plot highlights the monotonically decreasing fitness of the best position and contrasts the variation in the current positions fitness as the particle explores many of the local solution spaces.

WSS by looking at a sliding window of the data to observe whether the mean and variance are constant over time [Kins09] [ScKi11a].

Figures 4.12-4.13 show the first two moments of the velocity time series extracted in the previous section for the Sphere and Rastrigin function respectively. These are broken down into the transient and steady state portions of each of the trajectories studied. The circles indicate the moment calculated over a window of 32 and 128 elements respectively and the line is added to help visualize the trends in the moments. The sliding window size is selected such that it has a minimum of 30 elements needed for statistical significance and the size of the main features seen in the envelope of the velocity plots. During the transient, the figures show a WSS signal following an initial increase to maximum speed. In contrast, the steady state portion of the trajectory shows a constant mean but the variance expands over time. This confirms the visual inspection findings of Sec. 4.1.4.1 that the steady state portion is a heteroskedastic signal. Such behaviour can indicate the particles have converged so there are no social influences and are now exploring the parameter space based on the stochastic weights.

Also in the time domain, the auto-correlation of a signal is important to evolutionary algorithms as it reveals the long-term dependancies that depict the evolving processes [ScKi11a]. These features verify the search is not random and that the particles are indeed moving towards an optimum solution.

Figures 4.14(a) and 4.15(a) show the correlation for the particles trajectory along one dimension for the transient for the Sphere and Rastrigin functions. The remaining plots are found in Appendix A. The graphs show a strong correlation for the first few iterations consistent with the portion of the time series that maximizes the velocity as expected from [KeEb01]. However, the graphs do not show a very noticeable long term correlation. In the case of the Rastrigin function, the long term correlation is even less noticeable once

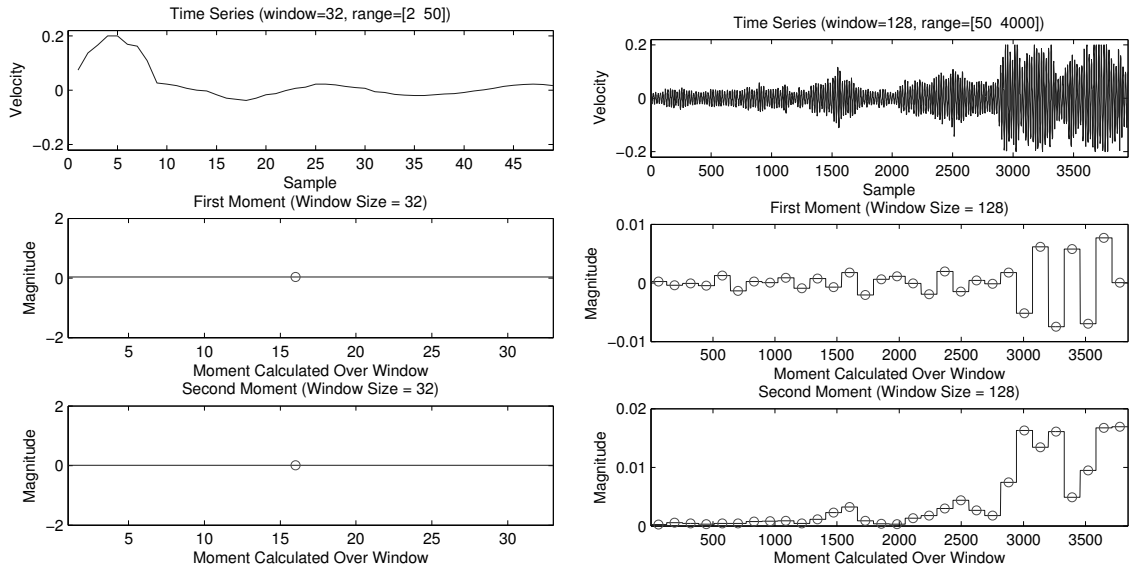


(a) Transient first and second moment for $G_1(\mathbf{x})$. (b) Steady state first and second moment for $G_1(\mathbf{x})$.

Fig. 4.12: First and second moment of particle trajectory on the Sphere function. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory.

the particle reaches the vicinity of the solution and is trapped within one of the cones that make up the parameter space. At this point, the correlation decreases as the particle is forced to switch directions many times. Figures 4.14(b) and 4.15(b) show the correlation for the steady state portion of the time series. As evident from observing the envelope for the velocities shown, there is a correlation that drives the trajectories of the particles is stronger for short term changes in position, and then weakens for longer term relationships. Overall, the correlation is lower than that found in the transient because the particle has converged on a solution and is now exploring the parameter space instead of traveling towards a destination.

For comparison, fixing the weights $S_{\varphi 1}$ and $S_{\varphi 2}$ as suggested in some tests in [EbSh00] and [Trel03], the graphs might show a very high autocorrelation. In this case, rather than exploring the parameter state, the particles can enter a quasi-periodic state that does not



(a) Transient first and second moment for $G_3(\mathbf{x})$. (b) Steady state first and second moment for $G_3(\mathbf{x})$.

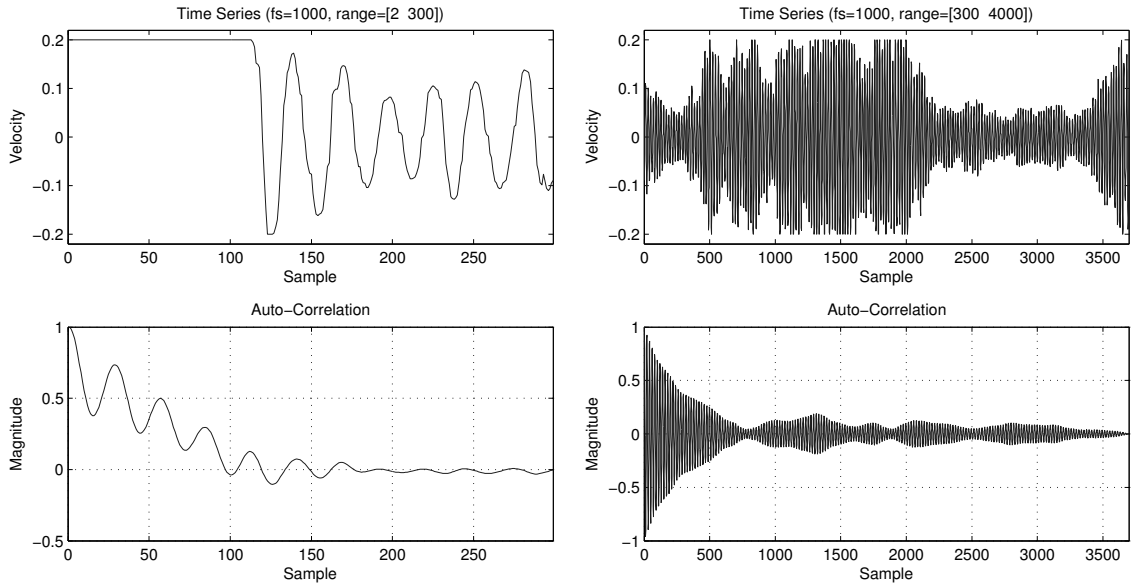
Fig. 4.13: First and second moment of particle trajectory on the Rastrigin function. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory.

help in finding the solution [ClKe02].

4.1.5.2 Frequency Domain Analysis

The power spectrum of the time series for the transient and steady state portions are computed to confirm the correlation analysis from the time domain and also gauge the amount of exploration in the signals of interest. The power spectrum for the transient and steady states for the same particle used throughout this paper are plotted in Figs. 4.16 and 4.17 for both the transient and steady state. A sampling or reference frequency of $f_s = 1000$ Hz is used throughout. The spectrums for the steady state show a peak at 500 Hz that is a side effect of the sampling frequency selected.

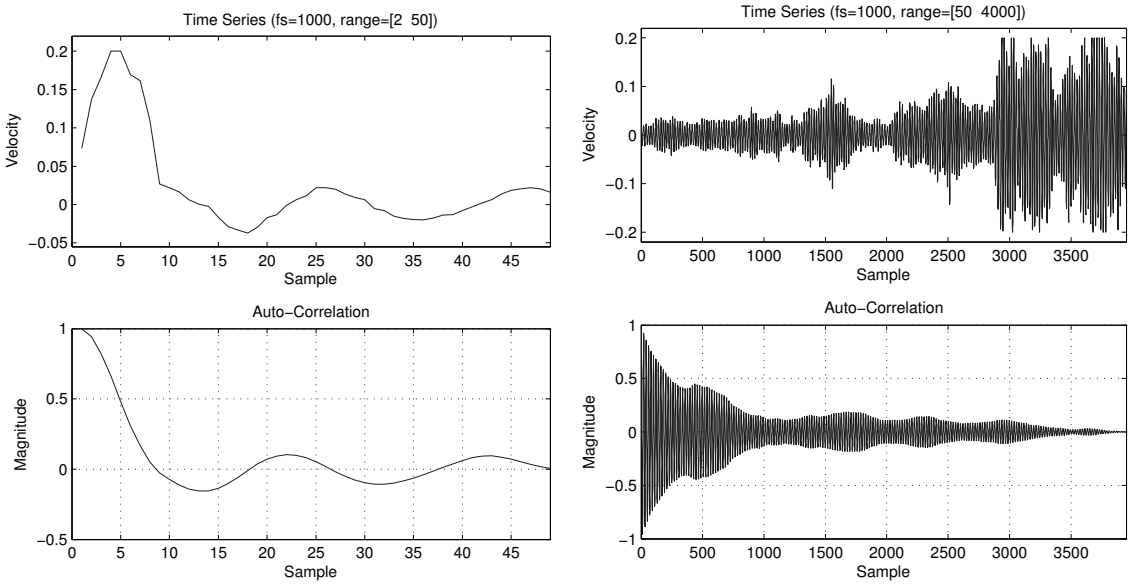
Intuitively, the plots agree with the observations from previous sections. During the



(a) Auto-correlation of trajectory for $G_1(\mathbf{x})$. (b) Auto-correlation of trajectory for $G_1(\mathbf{x})$.

Fig. 4.14: Auto-correlation of trajectory on the Sphere function. (a) Auto-correlation for transient portion of the trajectory. (b) Auto-correlation for the steady state portion of the trajectory.

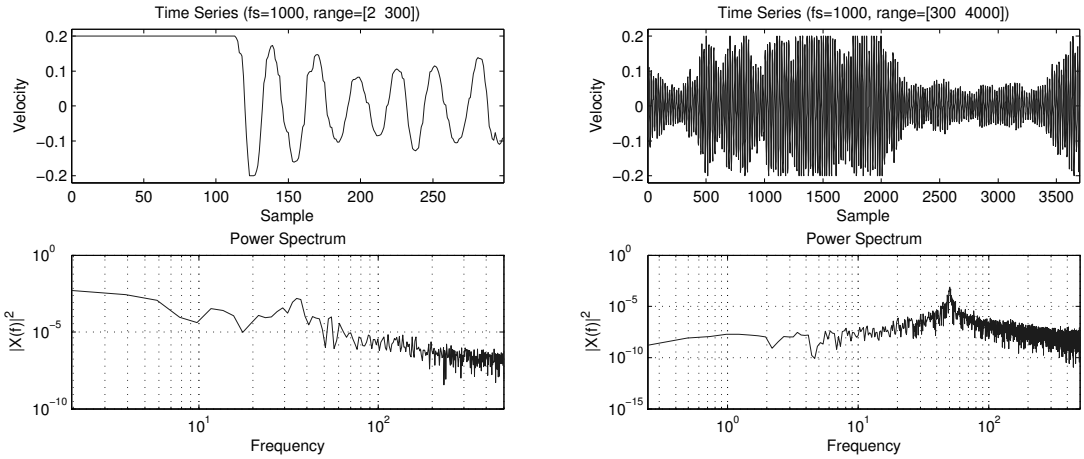
transient, the overall shape of the spectrum appears to have a negative slope with some variation around certain frequencies that correspond to the amplitude modulation of the overall signal. The steepness of the slope is determined with the path taken by the particles. When a particle is the best in the neighbourhood in the Rastrigin function, the particle begins exploring the parameter space locally and when some particles are optimizing the Rastrigin function and get trapped in a local solution during the transient, then the steepness of the slopes is reduced. In the first case this is because the particle is exploring the parameter space and without an external force telling it in which general direction to travel. The latter case exhibits a higher slope because the particle requires more energy to escape these local solutions. That is, the weights during some iterations may not be sufficient to escape the local minimum and therefore, small steps are taken to escape that space.



(a) Auto-correlation of trajectory for $G_3(\mathbf{x})$. (b) Auto-correlation of trajectory for $G_3(\mathbf{x})$.

Fig. 4.15: Auto-correlation of trajectory on the Rastrigin function. (a) Auto-correlation for transient portion of the trajectory. (b) Auto-correlation for the steady state portion of the trajectory.

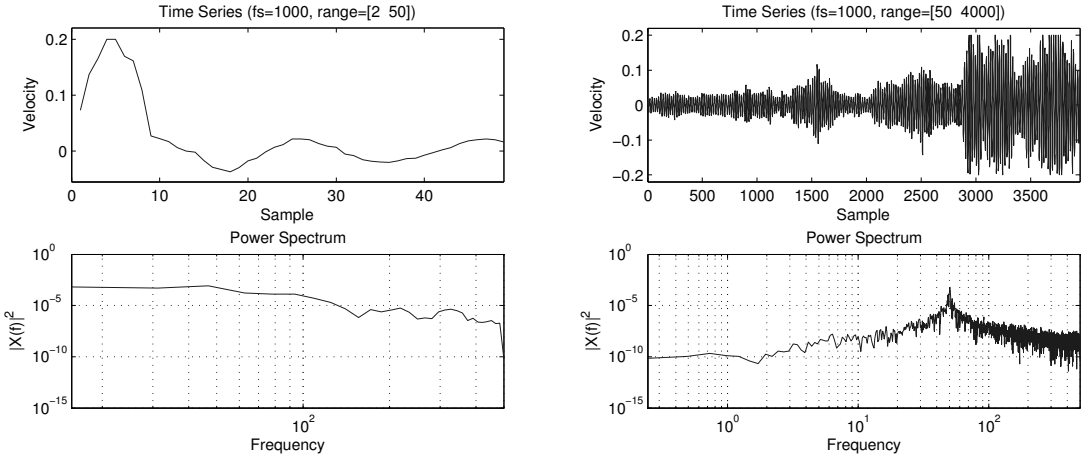
The negative slope shown in Figs. 4.16(b) and 4.17(b) are characteristic of self-affinity and long-term dependences. These show that the personal best solutions stored by each particle act as a memory for the system that enables particles to constantly move towards a better solution. For comparison, if the optimization algorithm selected new random solutions at each iteration with no influence from the past, then the slope would be equal to zero showing that there is no correlation from one iteration to the next (as seen in white noise in Fig. A.3(e)). At the other extreme, a very steep slope is indicative of very long-term correlated trajectories typical of a steepest decent algorithm with added randomness to explore the path on a way towards a solution (as seen in brown noise like Fig. A.3(f) or more pronounced in deep black noise). Thus, the balanced mixture of random behaviour with long memory is the specific behaviour that makes PSO suitable for cognitive machines.



(a) Power spectrum of trajectory for $G_1(\mathbf{x})$.

(b) Power spectrum of trajectory for $G_1(\mathbf{x})$.

Fig. 4.16: Power spectrum of trajectory on the Sphere function. (a) Power spectrum for transient portion of the trajectory. (b) Power spectrum for the steady state portion of the trajectory.



(a) Power spectrum of trajectory for $G_3(\mathbf{x})$.

(b) Power spectrum of trajectory for $G_3(\mathbf{x})$.

Fig. 4.17: Power spectrum of trajectory on the Rastrigin function. (a) Power spectrum for transient portion of the trajectory. (b) Power spectrum for the steady state portion of the trajectory.

4.2 Design of Scheduling Algorithm

Before designing the scheduler algorithm it is important to define the key constraints and assumptions used in the study. Then, the PSO-based scheduler can be designed through an encoding methodology and a cost function that links the tasks definitions to the optimization algorithm.

4.2.1 Assumptions and Constraints for the Scheduler

Given the use of EAs for the scheduler, then a *hybrid* configuration is selected. This is the most suitable option for spacecrafts as it minimizes the amount of resources used for the scheduler and allows the system to only evaluate new schedules if the system changes state. For example, in the *University of Manitoba Space Applications and Technology Society* (UMSATS) TSat nanosatellite, the states for the spacecraft are defined as shown in Fig. 4.18 and described in [KSFC12]. In this project, the spacecraft was designed to be in a mode for prolonged periods of time and switch upon commands or extreme anomalies. These type of state changes can be set to trigger a new schedule to be computed.

Furthermore, for a scheduler algorithm to be useful, it must be easily adaptable to various types of spacecrafts. Therefore, to facilitate this, the cost function is based on simple principles with cumulative penalties for violating different types of constraints. As such, the baseline algorithm is defined based on the following assumptions:

1. **No processor affinity** – This feature can be added in the cost function by incorporating penalties for violating processor affinity constraints.

2. **No preemption** – The use of preemption makes for more possibilities in SMP schedulers and thus makes it more difficult for EAs to find a solution. Schedules implementing preemption often define set rules for selecting tasks like RM. Furthermore, to keep over-

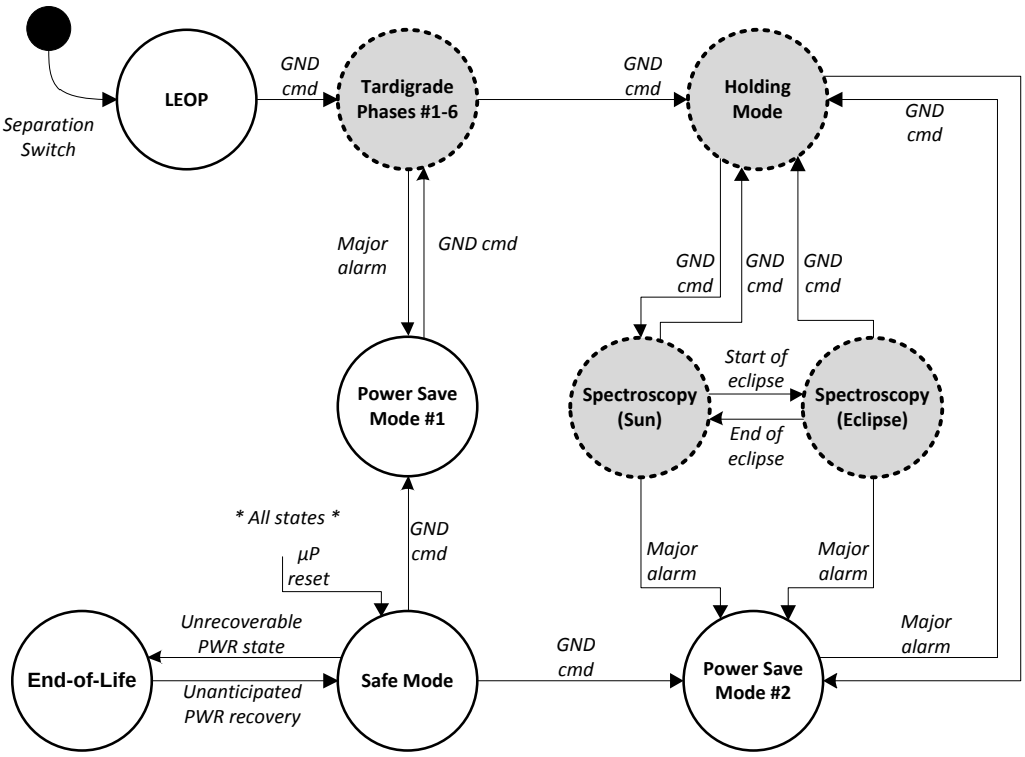


Fig. 4.18: Modes of operation for TSat nanosatellite. Each state requires a uniquely defined schedule to operate the various subsystems and tasks running at that time. This requires adaptive schedules for emergency modes to avoid getting stuck trying to execute non-operational subsystems. The states highlighted with dotted borders are those corresponding to payload operations [KSFC12].

head to a minimum, it is better to assign tasks to cores that can execute the task to completion.

3. **No migration** – This follows from the previous point that overcomplicates the design of a scheduler if one adds additional options to migrate tasks to other processors in a real-time system.

4. **No task precedence** – Predefined orders for tasks can be incorporated through careful definition of task properties (i.e., set T_S and T_d for tasks to enforce certain precedence requirements using very tight bounds) or by incorporating additional penalties in the

cost function for the scheduler whenever the precedence is not observed.

5. **No task priority** – Again, this can be incorporated through penalties on the scheduler.
6. **Tasks are aperiodic** – Tasks are treated as aperiodic during testing. Periodicity can be simulated through careful definition of tasks (i.e., define multiple instances at the required intervals).
7. **Task computation time** – The worst case computation time for all tasks is known *a priori* and includes the setup times necessary to activate the task.

4.2.2 Design of Encoding Mechanism

Preserving the evolutionary nature of PSO in the continuous domain requires the problem to be encoded to match discrete states in scheduling of tasks (order and processor selection) to the continuous position of tasks. There are many different methodologies for encoding tasks in EA-based schedulers. For example, [Cook10] randomly selects a task and moves the tasks in one direction (up, down, left, right) relative to the position of the task with respect to the task timing diagram. This method creates small perturbations that allow algorithms to search the local parameter space, however, the random perturbation, although well suited for SA, removes the emerging nature found in population-based EAs like PSO.

The encoding methodology selected for this thesis is based on the *random-keys* (RK) proposed by James Bean [Bean94] and tested for single machine scheduling problems in [CaEC07]. This encoding mechanism was originally designed to produce valid solutions to permutation problems using genetic algorithms without the need for problem specific representations of solutions to manage the crossover operators [Bean94]. The general idea of RK is that the EA would search a continuous parameter space that acts as a surrogate

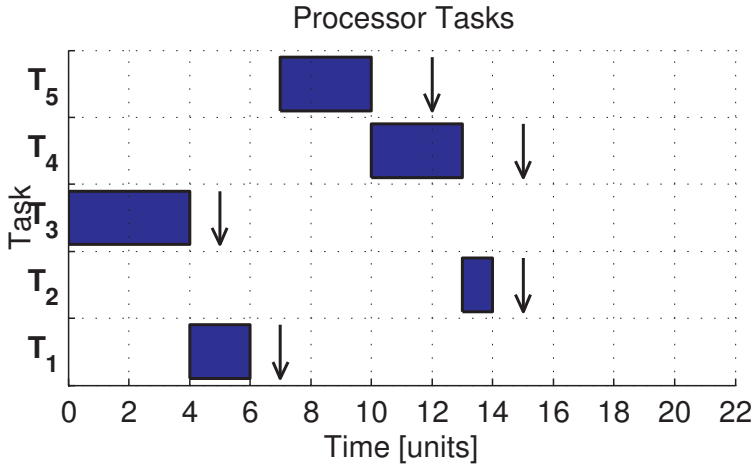
for the problem space (in this case the task allocation space).

In the context of scheduling for uniprocessor systems, the RK encoding maps each dimension to a task, such that each particle contains a full candidate solution. The mapping procedure is based on the dimension for each element in the $\mathbf{S}_x$ vector ($d = 1 \rightarrow T_1, d = 2 \rightarrow T_2, \ldots$). The sequence of tasks for a solution are obtained by sorting the position vector in ascending order thus giving the new order for the tasks [Bean94]. Since this relies on very close particle initializations such that they can exchange positions to produce the permutations, Bean suggests setting $S_{Xmin} = 0$ and $S_{Xmax} = 1$ for single processors. For example, given a set of 5 tasks shown in Fig. 4.19(a), a particle S_k is position at $\mathbf{S}_{x,k}(n) = \{0.1, 0.5, 0.3, 0.2, 0.4\}$ at time n corresponding to the schedule shown in Eq. 4.5a and visualized in Fig. 4.19(b). One iteration later, the particle moves to $\mathbf{S}_{x,k}(n+1) = \{0.2, 0.5, 0.1, 0.4, 0.3\}$ and finds an optimal schedule described through Eq. 4.5b and Fig. 4.19(c).

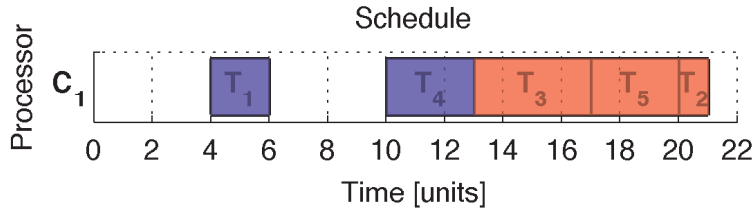
$$\mathbf{S}_{x,k}(n) = \Big\{ \underbrace{0.1}_{T_1}, \underbrace{0.5}_{T_2}, \underbrace{0.3}_{T_3}, \underbrace{0.2}_{T_4}, \underbrace{0.4}_{T_5} \Big\} \xrightarrow[\text{See Fig. 4.19(b)}]{\text{Produces schedule}} \Big\{ T_1, T_4, T_3, T_5, T_2 \Big\} \quad (4.5a)$$

$$\mathbf{S}_{x,k}(n+1) = \Big\{ \underbrace{0.2}_{T_1}, \underbrace{0.5}_{T_2}, \underbrace{0.1}_{T_3}, \underbrace{0.4}_{T_4}, \underbrace{0.3}_{T_5} \Big\} \xrightarrow[\text{See Fig. 4.19(c)}]{\text{Produces schedule}} \Big\{ T_3, T_1, T_5, T_4, T_2 \Big\} \quad (4.5b)$$

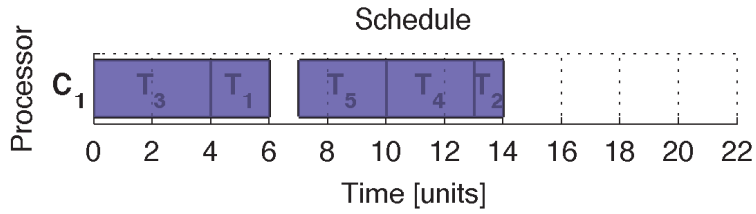
Extending RK to a C_N processors is accomplished by increasing the domain at initialization to $S_{Xmin} = 0$ and $S_{Xmax} = C_N$. Then, the mapping process first sorts the tasks and interprets the integer portion as the processor assignment and with the fractional portion as the task order [Bean94]. For example, given a set of 5 tasks shown in Fig. 4.20(a). A particle S_k is position at $\mathbf{S}_{x,k}(n) = \{0.1, 0.5, 0.3, 0.2, 0.4\}$ at time n corresponding to the schedule shown in Eq. 4.6a and visualized in Fig. 4.20(b). One iteration later, the particle moves to $\mathbf{S}_{x,k}(n+1) = \{0.2, 0.5, 0.1, 0.4, 0.3\}$ and finds an optimal schedule described



(a) Sample tasks to demonstrate RK encoding used in PSO.



(b) Mapping particle $\mathbf{S}_{x,k}(n) = \{0.1, 0.5, 0.3, 0.2, 0.4\}$ to a schedule which shows missed deadlines for 3 tasks (T_3, T_5, and T_2).



(c) Mapping particle $\mathbf{S}_{x,k}(n+1) = \{0.2, 0.5, 0.1, 0.4, 0.3\}$ to a schedule.

Fig. 4.19: RK encoding to map particle positions to a candidate schedule. (a) Shows 5 sample tasks used to demonstrate the encoding. (b) Shows the schedule produced by a particle at position $\mathbf{S}_{x,k}(n) = \{0.1, 0.5, 0.3, 0.2, 0.4\}$. This shows 3 missed deadlines. Note that for candidate schedules to be valid, they cannot violate the start time of the tasks, so there are 4 units of idle time between T_1 and T_4. (c) Schedule produced by a particle one iteration later when it moves to $\mathbf{S}_{x,k}(n+1) = \{0.2, 0.5, 0.1, 0.4, 0.3\}$. This time, the schedule produced is valid for the given set of tasks and does not have any missed deadlines.

through Eq. 4.6b and Fig. 4.20(c).

$$\mathbf{S}_{x,k}(n) = \Big\{ \underbrace{1.1}_{T_1}, \underbrace{1.3}_{T_2}, \underbrace{0.3}_{T_3}, \underbrace{0.2}_{T_4}, \underbrace{0.2}_{T_5} \Big\} \xrightarrow[\text{See Fig. 4.20(b)}]{\text{Produces schedule}} \left\{ \begin{array}{lll} T_1, & T_2 & \\ T_4, & T_5, & T_3 \end{array} \right\} \tag{4.6a}$$

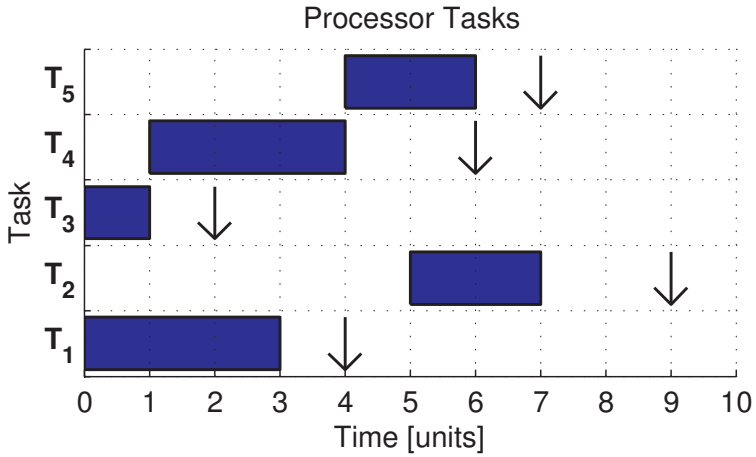
$$\mathbf{S}_{x,k}(n+1) = \Big\{ \underbrace{1.2}_{T_1}, \underbrace{1.3}_{T_2}, \underbrace{1.1}_{T_3}, \underbrace{0.1}_{T_4}, \underbrace{0.2}_{T_5} \Big\} \xrightarrow[\text{See Fig. 4.20(c)}]{\text{Produces schedule}} \left\{ \begin{array}{lll} T_3, & T_1, & T_2 \\ T_4, & T_5 & \end{array} \right\} \tag{4.6b}$$

The RK encoding enables the continuous PSO to be use for discrete problems. In doing so, it suffers from a major problem that many different solutions produce the same encoding of tasks [CaEC07]. Suggestions on how to overcome this are presented in Sec. 5.3 as they relate to the specifics of the cost function implemented in this thesis.

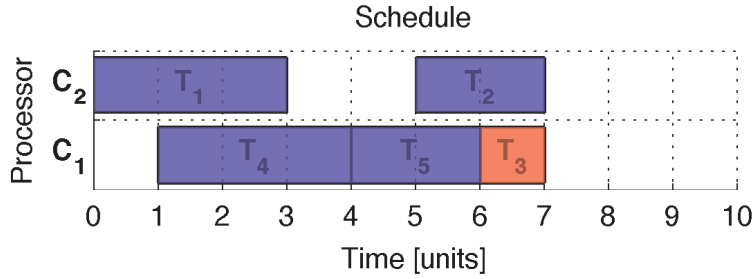
4.2.3 Design of Fitness Function

As described in Sec. 4.2.1, the objective is to design a generic cost function for scheduling where priorities, processor affinities, and other constraints could be incorporated easily. This provides the most flexibility in the design and use of the algorithm beyond spacecrafts. To accomplish this, the cost function starts with a modified *minimum-total-tardiness* (MTT) baseline and then cumulatively adds penalties for other constraints not met by the candidate solution.

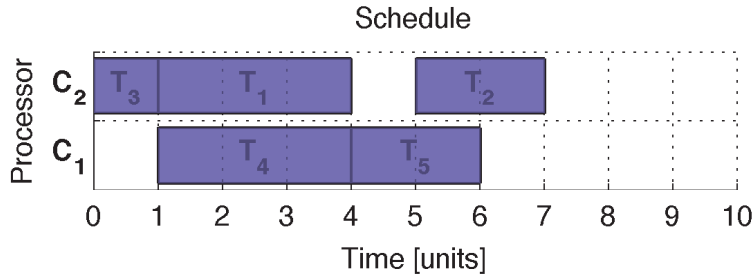
The MTT measures how late a task completes its execution with respect to its start time. This measure has been used in non real-time systems where the deadlines are an objective rather than a constrained. Normally, the tardiness is zero if the task completes before its deadline [Liu00]. In this thesis, this measure is augmented so that the tardiness is zero if a task starts executing at $t = T_S$, but it increments linearly for every unit of time that passes between the start time and the effective start time. In the context of hard real-time



(a) Sample tasks to demonstrate RK encoding used in PSO.



(b) Mapping particle $\mathbf{S}_{x,k}(n) = \{1.1, 1.3, 0.3, 0.2, 0.2\}$ to a schedule which shows missed deadlines for 1 tasks (T_3).



(c) Mapping particle $\mathbf{S}_{x,k}(n+1) = \{1.2, 1.3, 1.1, 0.1, 0.2\}$ to a schedule.

Fig. 4.20: RK encoding to map particle positions to a candidate schedule. (a) Shows 5 sample tasks used to demonstrate the encoding. (b) Shows the schedule produced by a particle at position $\mathbf{S}_{x,k}(n) = \{1.1, 1.3, 0.3, 0.2, 0.2\}$. This shows 1 missed deadlines. (c) Schedule produced by a particle one iteration later when it moves to $\mathbf{S}_{x,k}(n+1) = \{1.2, 1.3, 1.1, 0.1, 0.2\}$. This time, the schedule produced is valid for the given set of tasks and does not have any missed deadlines.

evolutionary schedules, MTT is used as a guide for the search. The costs associated with the tardiness is small to provide a small (gentle) slope with respect to zero (when produced by the unmodified MTT), that guides the search towards a possible solution. Although for some task distributions minimizing the tardiness can lead the search away from an optimal schedule, the cost difference for a tardy task is small such that the particles can escape that portion of the parameter space and explore alternative possibilities.

This fosters an overall faster response time for the entire system, while removing some of the plateau regions in the cost function that can hinder the execution of an EA. The modified MTT for a single task is defined by Eq. 4.7 and produces a number, $T_{tardiness}$, that indicates how late the task (mapped as d_index) started executing with respect to the current time unit, t_{p_index} of the processor p_index where the task is assigned. The maximum operator ensures that a tasks will not execute until its designated start time, T_S. If the current processor time, t_{p_index}, is less than the start time, T_S, then the processor will sit idle until such time the task can begin execution. The sum of the tardiness computed using Eq. 4.7 for each task adds up to the total cost associated with the schedule tardiness.

$$T_{tardiness} = \max\{0,\ t_{p\_index} - T_{d\_index,S}\} \tag{4.7}$$

In this thesis, the only penalties come from missed deadlines. A task missing a deadline is considered *late*, while *tardy* tasks start executing after their start time, but still meet their deadline. The initial design assigned a constant value for each penalty. A problem with a constant penalty was that it did not help the particles determine in which direction to move to improve a solution. A linear slope is used such that the penalty becomes more severe the further away the given task (decoded as T_{d_index}) was from meeting the deadline. This is very similar to the effect from the MTT, but is only employed on missed deadlines based on the current time and the computational time of the task. The penalty, $T_{penalty}$, is

defined as follows

$$T_{penalty} = |T_{d\_index,D} - t_{p\_index} - T_{d\_index,C}| \tag{4.8}$$

Equation 4.9 shows the combined $T_{tardiness}$ and $T_{penalty}$ that make up the fitness for the scheduler. More penalties can be added by implementing additional constraints on the algorithm. For example, task affinities can be evaluated during the encoding evaluation and assigned a value for tasks executed in a processor other than the desired core. This would further increase the fitness value of those candidate solutions, such that the ideal scheduler may produce the smallest fitness value. The final cost calculated, G_{cost}, gives a value greater than or equal to zero for all schedules.

$$G_{cost} = T_{tardiness} + T_{penalty} \tag{4.9}$$

4.2.4 Design of Cost Function Algorithm

Combining the encoding from Sec. 4.2.2 and the parameters for the fitness function from Sec. 4.2.3, it is possible to define a single function to evaluate the cost of a candidate solution produced by the PSO based scheduler (as shown in Algorithm 4.3). The input parameters for the scheduler cost function are a set of tasks, T_{set}, the position of a given particle, $\mathbf{S}_x$, and the number of processors in the system, C_N. This computes the fitness of the given candidate solution produced by PSO. In doing so, the algorithm recreates a full schedule allocation and computes the tardiness and penalties associated. The cost function can be divided into two portions (i) setting up the necessary variables (Lines 1-18) and (ii) computing the associated costs for each task (Lines 19-26).

The setup portion of the cost function defined by Algorithm 4.3 defines a number of local variables used to help decode the particle position into a schedule and allocate it to the appropriate processor. The current time for a given processor in the reconstructed schedule

is saved in t_{p_index}. This variable allows each processor to update its time based on the queue of tasks assigned to it and also helps to ensure that no tasks are executed before their designated start time. Lines 10-15 extract the task index from the RK encoding as described in Sec. 4.2.2, by finding the index of the smallest dimension on the position vector. Using that information, the processor index, p_index, is identified in Lines 16-17. This also maps particles that escape the S_{Xmin}-S_{Xmax} range by mapping anything smaller than S_{Xmin} to C_1 and everything above S_{Xmax} to C_N. This unconstrained approach is preferred to constraining the position of the particles because it allows the swarm to explore the space and naturally steer particles back towards the designated domain. Line 18 removes the allocated dimension by setting it to a very large value (practically selected as the maximum value for the decimal representation selected). Alternative methods of removing the particle from the list can improve the computational complexity of the cost function, but require more complicated data structures with dynamic memory allocation that can have similar performance effects to the simpler solution utilized.

To compute the cost, the function first establishes the effective start time for the task by evaluating the designated start time, T_S, versus the current time in the processor, t_{p_index}. As previously described, this enforces the constraint for the start time of each task. Then, the penalty is calculated as the difference between the absolute deadline for the task, T_D and the time the task would end executing. This penalty is only added if the task misses a deadline. Finally, the tardiness is calculated as per Eq. 4.7. Before adding the cumulative cost for that task, the time in the processor is increased for the next iteration.

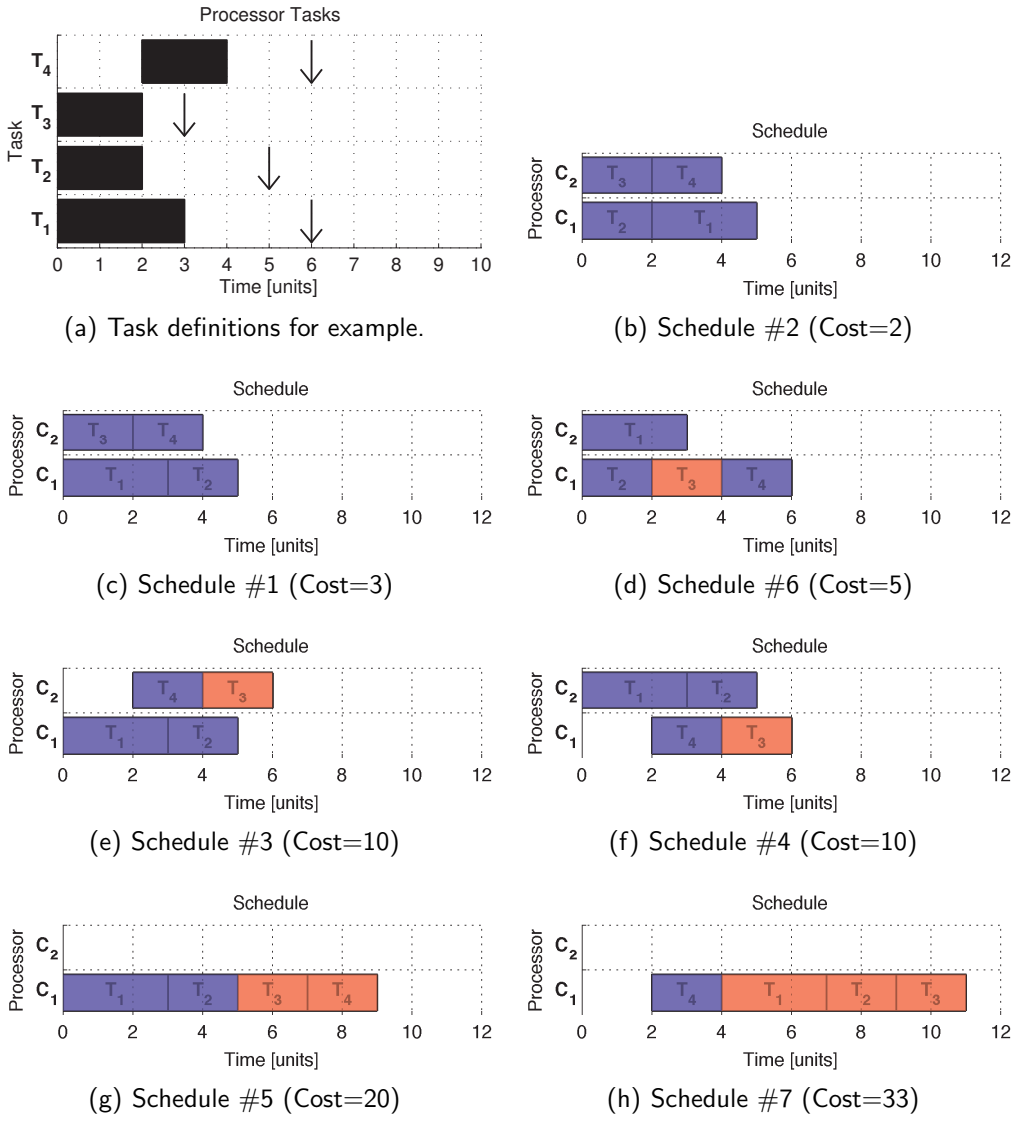
4.2.5 Example Cost Function Evaluations

Figure 4.21 shows a sample set of tasks with different schedules showing their associated costs. This example uses the tasks defined in Fig. 4.21(a) as the basis for demonstrating

Algorithm 4.3 Scheduler Cost Function, $G_{scheduler}(\mathbf{S}_{x,k}, T_\alpha, C_N)$

Ensure: $S_{x,k}$, T_α and C_N are provided.

1: **for** $p\_index = 1$ to C_N **do**
2: $t_{p\_index} \leftarrow 0$
3: **end for**
4: $G_{cost} \leftarrow 0$
5: $T_{tardiness} \leftarrow 0$
6: $T_{penalty} \leftarrow 0$
7: $p\_index \leftarrow 0$
8: $d\_index \leftarrow 0$
9: **for** $d = 1$ to D **do**
10: $d\_index \leftarrow 1$
11: **for** $d' = 1$ to D **do**
12: **if** $S_{x,k,d'} < S_{x,k,d\_index}$ **then**
13: $d\_index \leftarrow d'$
14: **end if**
15: **end for**
16: $p\_index \leftarrow \lceil S_{x,k,d\_index} \rceil$
17: $p\_index \in (1, C_N)$
18: $S_{x,k,d\_index} \leftarrow \infty$
19: $t_{p\_index} \leftarrow \max(T_{d\_index,S}, t_{p\_index})$
20: $T_{penalty} \leftarrow 0$
21: **if** $t_{p\_index} + T_{d\_index,C} > T_{d\_index,D}$ **then**
22: $T_{penalty} \leftarrow |T_{d\_index,D} - t_{p\_index} - T_{d\_index,C}|$
23: **end if**
24: $T_{tardiness} \leftarrow \max(0, t_{p\_index} - T_{d\_index,S})$
25: $t_{p\_index} \leftarrow t_{p\_index} + T_{d\_index,C}$
26: $G_{cost} = G_{cost} + T_{tardiness} + T_{penalty}$
27: **end for**
28: **return** G_{cost}



(a) Task definitions for example.

(b) Schedule #2 (Cost=2)

(c) Schedule #1 (Cost=3)

(d) Schedule #6 (Cost=5)

(e) Schedule #3 (Cost=10)

(f) Schedule #4 (Cost=10)

(g) Schedule #5 (Cost=20)

(h) Schedule #7 (Cost=33)

Fig. 4.21: Examples of cost function evaluations. (a) Set of sample tasks. (b) Optimal schedule given the set of tasks. (c) Valid, but not optimal, schedule for the given tasks. (d) Schedule showing one task missing a deadline. (e) Schedule showing that the start time for tasks is enforced. (f) Same as in (e), but with different processors to show the cost remains the same. (g) All tasks assigned to a single core. (h) Very bad schedule that missed three deadlines and has a high cost.

the increasing costs associated with worse schedule configurations.

To start, Fig. 4.21(b) and 4.21(c) both show valid schedules. The cost difference occurs because if T_1 is first, then T_2 starts executing further away from its T_S time than if the orders are reversed. This highlights the effect of the MTT portion of the cost function that rewards solutions that execute tasks closer to their start time. Note that when executing the full scheduler (as described in Sec. 5.3), if a valid solution is found, the optimization algorithm terminates even if that is not the optimal solution. If however, that termination criterion was not there, the optimization processes would continue searching for better solutions. In Fig. 4.21(d), T_3 starts executing 2 time units after its start time and also receives a 1 unit penalty for missing its deadline. This adds up to the majority of the cost so the 2 units associated with T_4 starting later become less important in rescheduling tasks. Figures 4.21(e) and 4.21(f) show that the cost is the same for a given task sequence regardless of which processor it is assigned to. This behaviour is required for an SMP scheduler as it provides more possible solutions. Note that in AMP with task affinities, such behaviour would require distinct costs to penalize any constraint violations. Figures 4.21(e), 4.21(f), and 4.21(h) show how the scheduler enforces the start time of a task even if it leaves idle units before. This is penalized by increasing the cost of the tasks that miss a deadline as a result of such operation. Finally, Fig. 4.21(g) and 4.21(h) show some worst case conditions to highlight the rapid increase in cost that is required in a cost function to accelerate the search. If the variations are very gentle (i.e., like in the Rosenbrock function), then the particles in the optimization algorithm may struggle to determine which direction to move in order to improve the solution. In general, the costs associated with the schedules may increase as more tasks are added to the system as both the tardiness and penalties grow with the problem size.

4.3 Summary

This chapter explored the characteristics of PSO through a novel data driven trajectory analysis that give insight into the behaviour of the algorithm. In addition, this chapter described the design of an encoding of solutions and cost functions to implement a PSO-based real-time scheduling algorithm for SMP systems. The next chapter describes the implementation and verification of the PSO algorithm and scheduler.

Chapter 5

System Implementation and Verification

The incremental implementation and verification consists of (i) the PSO algorithm, (ii) implementing the necessary analysis tools, (iii) the cost function and encoding for the scheduler, and finally, (iv) the scheduler simulator. Each portion is implemented and verified independently before proceeding to ensure that the dependancies are functioning as designed. This chapter describes the development environment and details of the software implementation.

5.1 Generic PSO Algorithm Implementation

Matlab was used throughout the thesis to implement the standard PSO algorithm (Algorithm 3.2) with many configurable parameters to enable testing different alternatives. The variations include the use of the inertia weight (S_ω), customizable topologies, and varying personal and social weights ($S_{\varphi 1}$ and $S_{\varphi 2}$). This enables different PSO configurations to be

tested that include the ones in this thesis as well as those described in [ScKA10], [ScKi11a], and [ScKi11b]. All this additional flexibility through customization extends the runtime of the algorithm. This is deemed acceptable as the run-time of the algorithm time is an implementation (both hardware and software) dependant measure which would render different results on a computer simulation from a real-time embedded processor. Furthermore, the metrics of interest are to be implementation independent as described in Sec. 4.1.1.

5.1.1 Customizable PSO Parameters

In order to control which version of the PSO algorithm is executed, the program loads a set of parameters from a configuration file in the form of a text document. The parameters with respect to the PSO implementation are divided into PSO parameters and cost function parameters. A full configuration example file is provided in Appendix C.1.

The PSO parameters define the particular variation of the algorithm executed. The number of particles, S_K, is flexible to be able to test for the minimum number of particles that provide good solutions for a given application, as that would reduce the computation complexity of the algorithm. The maximum number of iterations, N_{max}, is set to ensure the algorithm terminates even if no solutions were found. The neighbourhood topology is selected through a Matlab handle that allows the function to be called indirectly as if it was using an abstract function pointer with a predefined header (name, parameters, and return type). This handle takes in the set of particles, the current index S_k, and an optional parameter S_n defining the size of the neighbourhood. The function returns the index S_g of the best particle within the topology defined in the function. In the case of a gBest topology all the particles are evaluated, while for ℓBest topologies only the neighbourhood with respect to S_k is tested, as described in [ScKA10]. In a similar fashion, the personal and social weights are also computed in an external function that incorporates the uniform

random number generators from Eq. 3.2 and return a modified $S'_{\varphi 1}$ and $S'_{\varphi 2}$ that already incorporates the stochastic processes. This enables users to execute the algorithm with both fixed and stochastic weights. Finally, the inertia weight S_ω is defined through the $S_{\omega,max}$, $S_{\omega,min}$, and $S_{\omega,eph}$ parameters that can cause a linear decrease over the desired number of iterations. If a fixed S_ω is needed, the parameters can be defined as $S_{\omega,max} = S_{\omega,min} = C$ with $S_{\omega,eph} = 1$, where C is the desired constant inertia. Combining the S_ω with the corresponding definitions of the personal and social weights one can recreate the special cases of the constriction coefficient described in Sec. 3.2.2. The full constriction coefficient implementation is not used because there is less raw data available about the trajectories to reliably extrapolate the observations from the particle trajectories presented in the thesis to other PSO implementations.

The cost function is also defined using a Matlab handle to be able to utilize the same algorithm for many optimization problems. The initial implementation received a vector of position elements to compute the fitness. As the scheduler was developed, extra optional parameters for application specific information (i.e., the tasks and number of processors in the scheduling case) were incorporated. Three additional parameters directly linked to the parameter space described by the cost function are also included: S_{Xmin}, S_{Xmax}, and S_{Vmax}. These are used during the particle initialization and then in limiting the step size.

5.1.2 Saving Trajectories for Analysis

In addition to performing the optimization, the PSO algorithm has added features to save the trajectories of all the particles for analysis. This feature can be enabled through the configuration file in order to remove the unnecessary delays associated with constant writing to files. Each particle is stored in a separate file formatted as described by Fig. 5.1 through

a diagram representation. In essence, each line represents one iteration and consists of the position, velocity, and best position along each dimension, followed by the position fitness, best position fitness, and the index of the neighbourhood best. Note that the same results could be obtained by recomputing some parameters from the position and best position only. Since the information is available at runtime, it is easier to store all the parameters without a significanct expense in memory usage. Each variable is stored with six digits of decimal precision that provides high resolution for both comparing to the literature as well as for the analysis. This makes the trajectory of a particle when $D = 2$ and $N_{max} = 4000$ be approximately 344 KB, thus totalling 6.9 MB for a $S_K = 20$ particle set.

$S_{x,k,1}$ $S_{v,k,1}$ $S_{p,k,1}$ $\ldots$ $S_{x,k,D}$ $S_{v,k,D}$ $S_{p,k,D}$ $G(\mathbf{S}_{x,k})$ $G(\mathbf{S}_{p,k})$ S_g

Fig. 5.1: Structure of trajectory file for the S_k particle.

5.2 Time Series Analysis

The WSS, auto-correlation, and power spectrum analyses described in Sec. 4.1.5.1 are each implemented as a separate function that take in a time series and some additional parameters to produce the plots analyzed in this thesis. Before testing the functions on real data collected from running the PSO algorithm, each function implementation was validated against a known set of waveforms consisting of (i) a single sinusoidal, (ii) a composite signal with two sinusoidal terms, (iii) a square wave - generated using the `square` function, (iv) a triangle wave - generated using the `sawtooth` function, (v) white noise - drawn from a random normal distribution using `randn`, and (vi) brown noise - generated using the synthesis fractional Brownian motion algorithm available through `wfbm` (in this case with the Hurst exponent set to 0.5). The results of the verification procedures are presented in Apprendix A.1.

The WSS analysis computed the first two moments of a given time series using a sliding window size provided as an input argument. The moments are computed using the Matlab built-in `moment` function. The plots for the the verification showed that the periodic signals are WSS for window sizes of 256, while the brownian motion is non-stationary as expected. The results of these verification tests are provided in Appendix A.1.1.

The auto-correlation of the time series are evaluated using the `xcorr` function in Matlab. As expected, this shows that the periodic signals have a long correlation, while white noise is not correlated at all. Finally, the brownian motion reveals the anticipated short term correlations compared to the periodic signals. The plots showing the auto-correlation of these test functions are included in Appendix A.1.2.

Finally, the power spectrum is computed by squaring the result of the Fast Fourier Transform built into Matlab, `fft` and plotting the magnitude versus the frequency. In this case, a sample frequency is required as a reference and it is fixed at $f_s = 2000\ Hz$ during the verification. The results show peaks at the corresponding frequencies for the sinusoidal and composite signals. The square and triangle waves shows broadband spectra with peaks corresponding to the main coefficients evaluated. The time series containing white noise shows a flat wide band spectrum, while the brownian motion shows a negative slope indicating the data is correlated as expected. These figures are included in Appendix A.1.3.

The trajectory analysis from Sec. 4.1.4 requires many runs of the algorithm using the same initial conditions. This process is automated through Matlab scripts that set the seed for the pseudorandom number generator at the beginning of each iteration to initialize the particles and then again changes it again based on the current run of the algorithm so that it would render different stochastic results for the PSO runs. It is important to note that the algorithm is limited to running approximately 100 runs before having to clear the Matlab cached variables. The reason for this was determined to be related to the maximum number

of file pointers that can Matlab retains until the memory is clear. If the number of particles increases from 20, then the maximum number of runs may decrease accordingly. This does not limit the analysis, but rather requires additional precautions to either segment the work or explicitly close and clear many file pointers.

5.3 Implementation of Scheduling Algorithm

The implementation of the scheduling algorithm includes adapting the Matlab PSO algorithm and implementing the cost function from Algorithm 4.3 in Matlab. The PSO algorithm implementation consists of an augmented version of the one used for the time series analysis that incorporates a new terminating condition, an additional perturbation that is directed to tasks stuck in local solutions, and a slight modifications to record the schedule at each iteration as a record that can be used during analysis.

The new termination criteria for the algorithm consists of either a valid schedule that meets all the requirements of the system or the maximum number of iterations being reached, N_{max}. That is, a schedule with no missed deadlines terminates the algorithm regardless of whether there is a slightly better solution. The difficulty is that as shown in Sec. 4.2.5, it is not possible to obtain enough information from the cost function to determine whether a valid schedule is indeed the optimal schedule for that set of tasks. Since the cost function already finds all the missed deadlines, this extension consists of merely returning that information to the PSO algorithm and adding a couple of extra lines to terminate if a valid schedule is found.

Furthermore, taking advantage of already available computed schedules, one can add a few lines of code to Algorithm 4.3 after Line 17 to add the current task to a queue to produce the schedule associated with that particle being evaluated. This information is further returned to PSO algorithm such that when recording each trajectory as per Sec. 5.1.2, one

can also store the associated task allocation in order to be able to examine the performance of the algorithm by visual inspection of the trajectory files for each particle.

Finally, as suggested by [CaEC07], using RK can generate for multiple solutions leading to the same schedule. Therefore, borrowing from SA, a mutation is implemented with a linearly decaying probability that can affect the dimension of a task that is missing a deadline. This directed approach improves on the work of [CaEC07] by directly focusing on critical tasks as opposed to randomly teleporting a particle as suggested in [BSDK13]. Although the mutation affects a single dimension, depending on the newly assigned value, that can cause the entire schedule to be recomputed. However, since the best position of the particle remains the same, solutions resulting from mutations that destroy the entire schedule instead of simply escaping local solutions vanish after a few iterations as the particle approaches its previous region of the parameter space. The effect of adding the mutation parameter helps to improve the overall performance of the algorithm for higher system loads as visible in Sec. 6.3 where the large variance in the iteration count (compared to where the median is) indicate that particles are successfully escaping local solutions.

5.4 Schedule Simulations

A simulation of the scheduler was built to help verify its operation and test the performance of the algorithm. The software has the option of reading in a user defined task set or generating a random one with certain properties. Then, the simulator schedule the given set of tasks a repeated number of times, logging its progress for analysis.

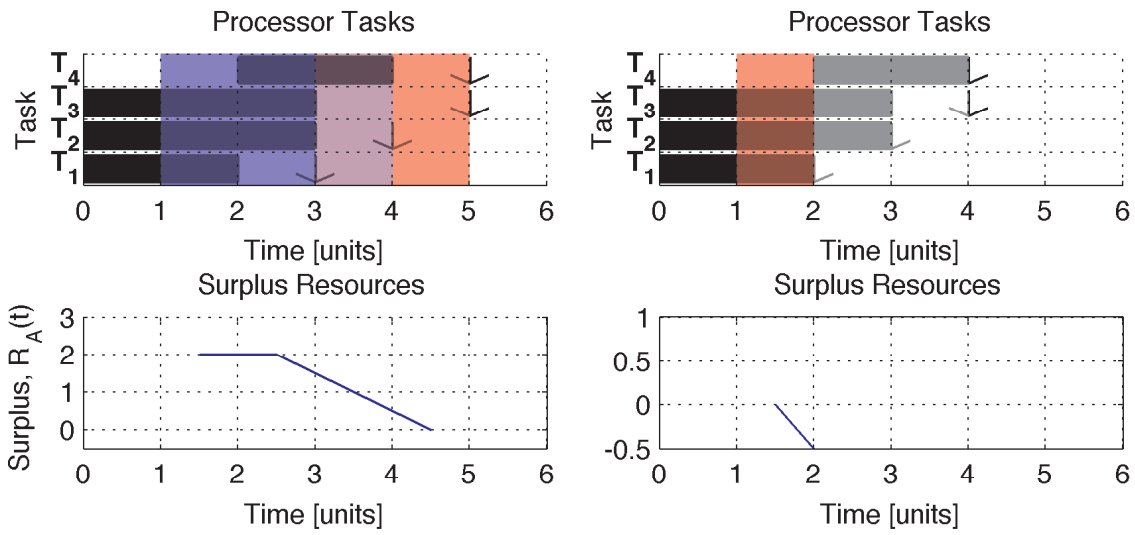
User defined tasks sets are used for testing the performance of the algorithm against some rudimentary conditions like those outlined in Sec. 4.2.1. To generate random tasks sets, the user defines six parameters that provide ample flexibility for testing different conditions. The user provides a range (minimum and maximum values) for T_S, T_C, and T_d

so that random values can be given limits that are representative of a particular application. Similarly to the topology selector, the distribution is customized via a Matlab handle to facilitate applying the same scheduler to different scenarios. This thesis uses a uniform random distribution, since this does not bias the tasks in any way. The implementation allows the use of other configurations, such as simulating arrival times based on a Poisson distribution. To ensure the task sets generated are scheduleable, Eq. 2.2 is computed for the tasks generated. Furthermore, the user defines a range in which the mean surplus processing power must reside. This metric ensures that the task are schedulable with an average surplus representative of the particular application being modelled.

In the process of generating random tasks, it is useful to visualize areas that can be troublesome for the algorithm based on the surplus resources at each iteration. To facilitate this, the task sets generated are plotted with a gradient background that indicates whether there are lots of available resources in blue and critical sections using a red background. This novel use of the schedulability test is demonstrated in Fig. 5.2(a). In this figure, there are three tasks with $T_S = 0$ for a $C_N = 2$ system, but all of the tasks have some slack between their completion time and the deadline. For example, this allows moving T_3's start time and still abide by all the real-time deadlines. As such, the background from $1 < t \leq 3$ is blue showing that although there are lots of tasks, they can be shifted somewhere. While, for $4 < t \leq 5$, the background is red because anything moved to this area would not have anywhere else to go. This is reflected in the actual values for $R_A(t)$ shown in the graph where there the surplus resources diminish to zero as there are no free resources.

In contrast, Fig. 5.2(b) shows a schedule where only T_3 has some slack and therefore, the number of available resources is negative. This is reflected by the white background on the tasks showing that they cannot be completed. Note that this metric is only defined for $t > 1$ as per Eq. 2.2. More complicated examples are provided in Sec. 6.3 and highlight cases where the surplus resources increase and diminish over time as would occur in many

applications.



(a) Sample task definition showing available resources through the blue-to-red gradient.

(b) Sample task definition showing negative resources through a white background.

Fig. 5.2: Visualization of task set surplus resources assuming $C_N = 2$. (a) Sample task definitions showing that there are some surplus resources for $t < 3$, then the resources diminish because the only way to schedule the tasks is to shift one of the tasks with $T_S = 0$ in time. (b) Sample tasks that cannot be scheduled due to the lack of available resources.

5.5 Summary

This chapter described the implementation of the PSO algorithm, the scheduler, and the simulator used to test the algorithm. This implementation is used in the next chapter to run three experiments designed to test the performance of the scheduler under different conditions.

Chapter 6

Experiments and Discussion of Results

The following experiments are designed to measure the performance and limitations of the scheduler cost function designed for the PSO algorithm. The experiments are organized for 1, 2, and 4 symmetric processors to show the limitations of the algorithm in each configuration. For each number of processors, there are three key experiments conducted: (i) benchmark – testing performance under extreme known conditions, (ii) load experiments – testing the scheduling algorithm performance under different system loads, and (v) scalability experiments – testing the algorithm with increasing number of tasks.

6.1 Experimental Setup

The software was developed and tested on Matlab version R2010b running on Mac OS X version 10.6.8 using 2×2.4 GHz Quad-Core Intel Xeon processors with 12 GB of 1066 MHz DDR3 memory. In addition, to expedite the testing process, multiple simula-

tions were performed simultaneously on four Mac Pro workstations running Matlab version R2012b under Mac OS X version 10.7.5 using 2×2.4 GHz Quad-Core Intel Xeon processors with 32 GB of 1066 MHz DDR3 memory. Regardless of the processing environment, the bottleneck for the simulations was saving the trajectories for analysis and the memory write speeds of the computers. These accounted for over 24 GB of data that is included in the DVDs accompanying this thesis and described in Appendix E.

6.2 Benchmark Experiment

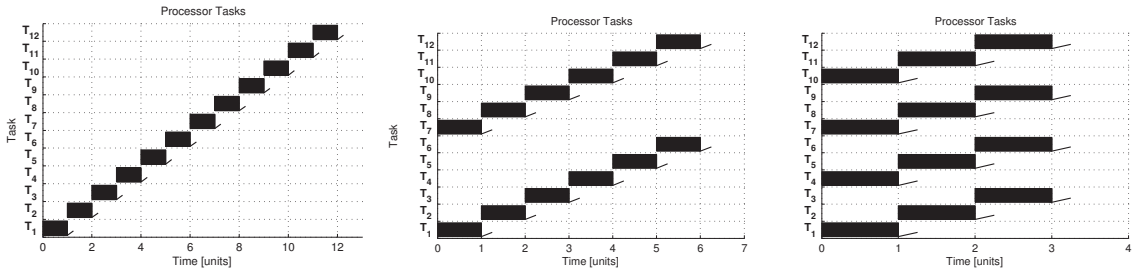
The benchmark example uses a set of tasks that maximize the system load, no surplus resources, and test the performance of the algorithm for small task sets. This is an unusual case that is not representative of real systems, but serves as a point of comparison that also tests the limitations of the system. The key parameters for the test are summarized in Table 6.1. Only $T_N = 12$ tasks are used to get a general reference point of how the system performs under extreme conditions. The number of particles is increased from 20 in the trajectory study in Sec. 4.1.4 to 30 to account for the increased dimensionality of the problem (increasing from $D = 2$ to $D = T_N = 12$). This number of particles is consistent with the suggested ranges as described in Sec. 3.2.1. The initial conditions for the particle positions are selected so that they would be uniformly distributed in the domain associated with each of the processors being used as per the RK encoding. Once running, the particles move between processors and can even escape the parameter space without affecting the performance. Similarly, the velocity is limited to allow particles to move from one processor to another and possibly outside the initial parameter space. The global topology is selected as it is widely used in most PSO implementations. The weights and inertia coefficient are selected to match the suggested values from Shi and Eberhart's study of particle convergence [EbSh00]. Finally, a linearly reducing mutation probability is

used to relocate some of the tasks missing deadlines at each iteration to help the algorithm escape some local solutions.

Table 6.1: Parameters for scheduler benchmark experiment.

Parameter	**Exp. #1a**	**Exp. #1b**	**Exp. #1c**
Number of processors, C_N	1	2	4
Number of tasks, T_N	12		
Number of dimensions, D	12		
Number of particles, S_K	30		
Position limits, S_{Xmin} and S_{Xmax}	[0, 1]	[0, 2]	[0, 4]
Velocity limits, S_{Vmin} and S_{Vmax}	[-1, 1]	[-2, 2]	[-4, 4]
Max. number of iterations, N_{max}	5,000		
Neighbourhood topology	gBest		
Personal weight, $S_{\varphi 1}$	1.49445 [EbSh00]		
Social weight, $S_{\varphi 2}$	1.49445 [EbSh00]		
Inertia weight, S_ω	Constant at 0.729 [EbSh00]		
Mutation probability	Linearly reduce $[0.4, 0.1]$		

The task sets for the benchmark experiments are evenly distributed through the processors used (12 tasks in one processor, 6 in each of the two processors, and 3 in each of the 4 processors). The computation time for all tasks is set to $T_C = 1$ with immediate deadlines of $T_d = 0$. The start times, T_S are arranged to maximize the system utilization. Fig. 6.1 shows the task definitions for each configuration tested. The uniprocessor configuration has a single global optimum, while for $C_N \geq 2$ there are many more options as there are multiple tasks with the same properties that can be scheduled in any of the processors. Alternative options for uniquely defined schedules that would produce unique solutions for multiprocessors were considered, but this approached the scenarios of task precedence that were not part of the objectives for this research.



(a) Definitions for $C_N = 1$ benchmark experiment.

(b) Definitions for $C_N = 2$ benchmark experiment.

(c) Definitions for $C_N = 4$ benchmark experiment.

Fig. 6.1: Benchmark experiment tasks definitions. The deadlines for each task occur immediately at $T_D = T_S + T_C$, thus reducing the laxity to zero for all cases. For $C_N \geq 2$ multiple tasks are defined for each time slot as shown in the subfigures. (a) $C_N = 1$, (b) $C_N = 2$, and (c) $C_N = 4$.

Overall, this configuration is designed to produce a minimum value of zero in the cost function, that helps test the algorithm, while also producing results that are intuitively perceived as valid or invalid. Conceptually, the sets of tasks described in Fig. 6.1 create a parameter space with many peaks when using the RK encoding. In the single processor case, a single global minimum exists and any task placed out of sequence would cause at least 1 missed deadline depending if it is placed at the end or whether it shifts all other tasks causing a chain of missed deadlines. In the multiprocessor setting, the peaks and valleys become more significant as there are now multiple solutions, so the parameter space approaches that of the Rastrigin function.

Each of the three scenarios is executed 50 times with random initial conditions for the particles each time. The performance is measured in terms of (i) whether a schedule is found, (ii) number of iterations until a solution is found, (iii) number of missed deadlines when the algorithm terminates (in case it reaches N_{max}), and (iv) the cost to provide a reference. The results of these scenarios are presented through a box plot in Fig. 6.2 and the results Table 6.2.

At first, it may appear counter-intuitive that the performance of the scheduler decreases

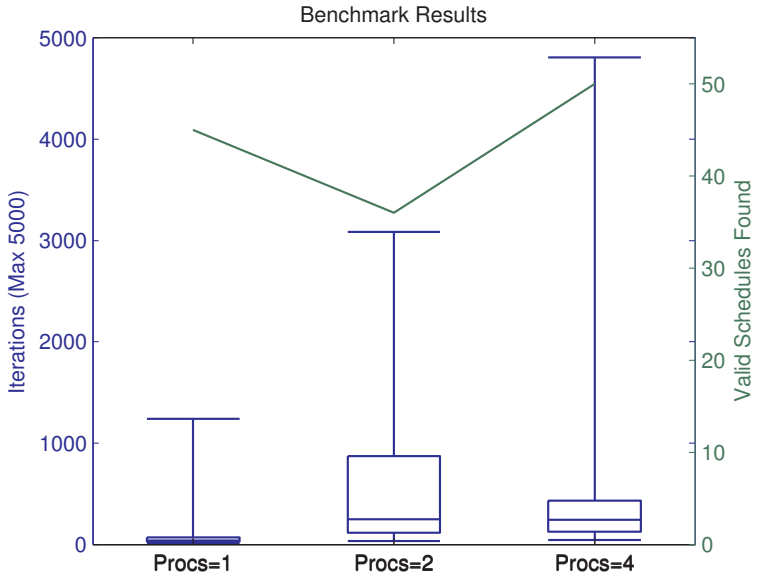


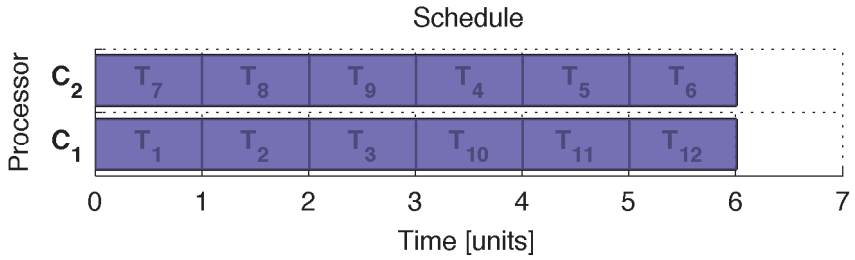
Fig. 6.2: Results of benchmark experiment. The box plot shows that the majority of the runs find a solution in under 1,000 iterations and have a long tail demonstrating the cases where the algorithm gets stuck in local solutions.

Table 6.2: Parameters for scheduler benchmark experiment.

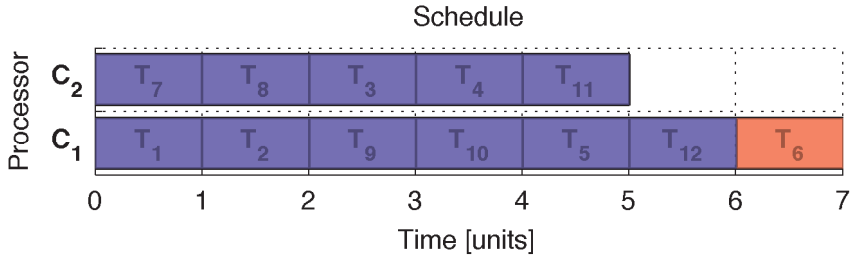
Test	Schedule	Runs	Iterations (Mdn)	Missed	Cost
$C_N = 1$	Valid	45/50	128.9 ± 250.3 (40.0)	0.0 ± 0.0	0.0 ± 0.0
	Invalid	05/50	$5,000.0 \pm 0.0$ (5,000.0)	1.0 ± 0.0	20.0 ± 5.5
$C_N = 2$	Valid	36/50	592.1 ± 743.1 (246.5)	0.0 ± 0.0	0.0 ± 0.0
	Invalid	14/50	$5,000.0 \pm 0.0$ (5,000.0)	1.7 ± 0.7	8.4 ± 4.6
$C_N = 4$	Valid	50/50	431.3 ± 739.9 (241.5)	0.0 ± 0.0	0.0 ± 0.0
	Invalid	00/50	N/A	N/A	N/A

for fewer processors. However, one must keep in mind that, for the task sets utilized, there are more valid combinations as the number of processor increase. Furthermore, the ramifications of a single task changing position are more significant for the single processor case because a single task can create a chain of events that raise the overall cost than in the multiprocessor through the cumulative penalties. For example, if task T_3 starts

at the end of the queue for a single processor and moves to the spot for T_9, the cost appears to increase because of all the penalties accumulated. As such, the scheduler has a difficult time moving the task incrementally towards the desired location. However, in the multiprocessor scenario, these variations are less significant because the number of tasks decreases per processor (hence the motivation for the system load experiment in Sec. 6.3). As an example, Fig. 6.3(a) and 6.3(b) show a valid and invalid schedule with their associated costs.



(a) Valid schedule with $cost = 0$.

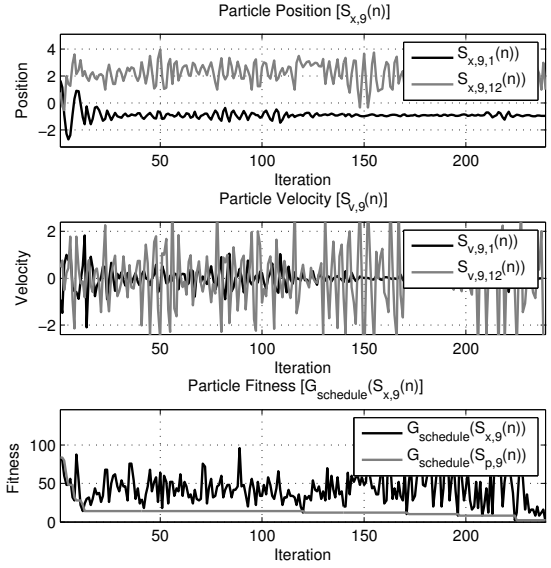


(b) Invalid schedule with $cost = 2$.

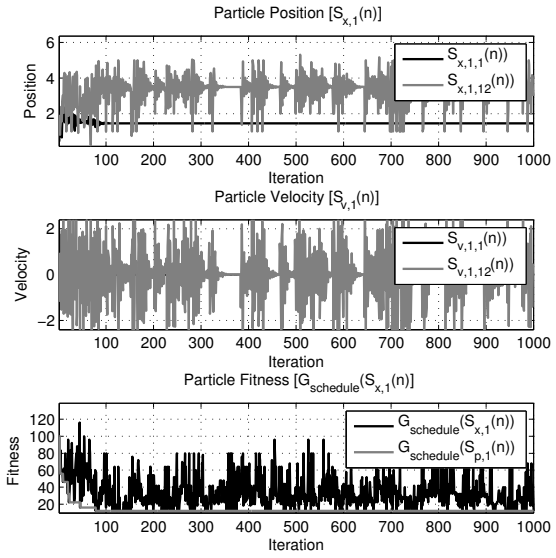
Fig. 6.3: Sample schedules generated for benchmark experiments for $C_N = 2$. (a) Valid schedule generated (test05-p2-t12-benchmark/008/). (b) Invalid schedule generated (test05-p2-t12-benchmark/005/). Note that in SMP there is no preference for tasks to be executed in a particular processor, so T_5 and T_{11} have different results in each run that are both valid.

The mean number of iterations for the 50 runs are significantly higher than the median calculated.This is because the algorithm has a fast transient towards the solution and then slows down for a local search. This is consistent with the number of missed tasks and associated fitness of the solutions produced. Furthermore, one can confirm the behaviour

through the trajectory of a particle, such as Fig. 6.4 obtained from a successful run for $C_N = 2$ versus an unsuccessful run for the same number of processors. In Fig. 6.4(a), one can see that the position tested various values for $d = 12$ corresponds to the task that should be towards the end of the schedule, so whenever the position decreases significantly, the associated cumulative cost spikes to force the particle to switch directions. This is reflected in the big envelope formed around the position that sees immediate spikes (usually cost through exploration or mutation) followed by an attenuated signal towards the optimal position.



(a) Trajectory for $S_k = 9$.



(b) Trajectory for $S_k = 1$.

Fig. 6.4: Sample trajectories from benchmark experiment. Both graphs show the $d = 1$ and $d = 12$ corresponding to tasks that should be scheduled at the beginning and end of the schedule respectively. (a) Results for successful run of the algorithm (test05-p2-t12-benchmark/008/p9.txt). (b) Results from unsuccessful run of the algorithm (test05-p2-t12-benchmark/004/p1.txt).

The full trajectory analysis is not performed as these are too short to perform the proposed techniques. Those become more valuable in subsequent experiments where both the transient and steady state behaviours are longer. In this case, Fig. 6.4(b) shows a very

rapid transient of less than 100 iterations before the local search. Note that slowing down the algorithm through a smaller S_{Vmax} as proposed in Sec. 4.1.4.2 would compromise the performance of the scheduling algorithm and in many cases produce worse results because the maximum step size would not allow particles to move from one processor to another very freely and thus would give many repeated task sequences as candidate solutions.

6.3 Load Experiments

In the load experiment, a random set of tasks is defined such that the overall system load approaches 10%, 50%, and 80%. Since there are no specific hardware processors associated with these tasks, the time is measured in units. Therefore, the system load is defined as the number of computational units over the number of time units from the beginning ($t = 0$) to the completion of the last task. As such, a 10% load indicates that the processors would be busy for 10% of the time and idle for the remaining time. Furthermore, the estimates are for overall system load, not for the individual processors. Therefore, as the number of processors grows, the system can take more tasks while retaining the same load. This approach was selected over a load shedding analysis because for distributing loads one must have increased flexibility with the tasks start times in order to not just move them to another processor, but to also reduce the overall completion time. Since in these experiments the task start time is well defined, the distribution of tasks is selected by the algorithm to meet the requirements and not to balance the system. Such constraints can be incorporated as an additional penalty before line 28 in Algorithm 4.3 by evaluating the computational time of each of the processor queues.

The selected values for the loads are modelled after analysis conducted at MAW and NASA. The MAW estimates are used with permission from the company. The 10% load is selected as it is roughly the estimated nominal load for the RCM spacecraft being de-

signed by MAW running 18 tasks in a single processor [MAW09]. Similarly, the 80% load constitutes the worst-case (and unrealistic) load for the RCM satellites rounded up to the nearest tenth [MAW09]. This extreme scenario happens when every task is executing even if it violates operational needs (i.e., concurrent payloads). The middle marker of 50% load is taken as the suggested value from the NASA Gold Standards for the estimated load at the preliminary design stage for a satellite [GOLD09]. Note that maximum 30% loads are recommended for the delivery phase of a satellite, however this is not tested as 10-50-80 experiments cover that range and give insight into the performance of the algorithm over a wider spectrum.

Table 6.3: Random task generator parameters for load experiment.

Experiment Run	T_S **Range**	$R_A(t)$ **Range**	**Calc. Sys. Load**
$C_N = 1$, Load= 10%	[0, 400]	[175, 200]	12.75%
$C_N = 1$, Load= 50%	[0, 100]	[30, 100]	55.96%
$C_N = 1$, Load= 80%	[0, 50]	[10, 100]	78.69%
$C_N = 2$, Load= 10%	[0, 500]	[250, 280]	10.04%
$C_N = 2$, Load= 50%	[0, 110]	[20, 35]	50.96%
$C_N = 2$, Load= 80%	[0, 22]	[10, 50]	78.13%
$C_N = 4$, Load= 10%	[0, 400]	[200, 250]	9.88%
$C_N = 4$, Load= 50%	[0, 20]	[10, 50]	50.00%
$C_N = 4$, Load= 80%	[0, 8]	[1, 50]	77.00%

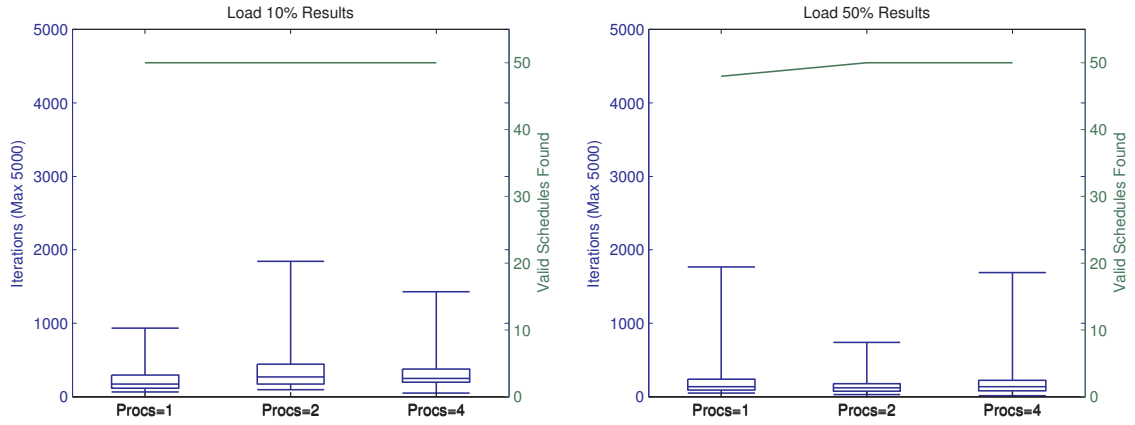
The tasks are generated randomly using the procedure described in Sec. 5.4. For this purpose, T_C is randomly selected from a uniform distribution between $[1, 4]$ and T_d is taken from a distribution between $[7, 14]$. This range of deadlines allows some tasks to have small variations in their position, while others can interchange positions without missing deadlines (thus creating more local solutions). In contrast with the benchmark experiment, the objective here is to see many local solutions with small variations in the cost to evaluate

how the algorithm performs to gentle slopes in the parameter space. The T_S and $R_A(t)$ vary for each test to adjust the overall system loads to the desired values as shown in Table 6.3. The ranges for each were established through trial and error to generate tasks sets with loads approximating the desired values. The $R_A(t)$ constraint ensures that valid schedules are produced and helps to distribute tasks such that for example, an 80% load does not contain an unschedulable cluster of tasks with similar start times and deadlines. The sample tasks sets for the three different loads using a two processor in Fig. 6.6-6.8. The remaining tasks sets are included in Appendix D.1.

The PSO parameters described in Table 6.1 for the benchmark experiment are maintained for this analysis. The only differences are that $T_N = D = 20$ and that for loads of 80% the maximum number of iterations is increased to $N_{max} = 10,000$. The increase is empirically derived to help gauge whether more iterations can help find the solution. The number of tasks is roughly the number of tasks utilized in the RCM mission [MAW09].

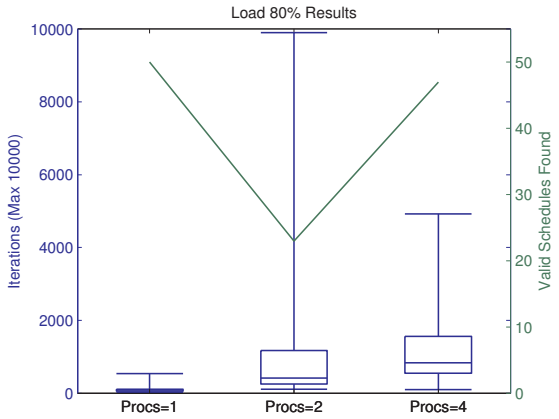
The results from 50 runs of each case are averaged in Fig. 6.5 and Table 6.4. The algorithm performs very well for all loads with different number of processors. Once again, there is a significant difference between the median and mean number of iterations required to find a solution. The mutations are helping some particles escape the local solutions and cause the large variances calculated, however, for the most part there are some areas of the parameter space where particles are trapped for long periods of time. Based on the large discrepency in mean and median, it is almost better to restart the algorithm after an empirically defined number of iterations with a different initial conditions to accelerate the search rather than attempt to escape a local solution.

One unexpected result was that for $C_N = 2$ with 80% loads, the performance dropped significantly with most particles producing schedules with a single deadline missed. This correlates with the $R_A(t)$ plot from Fig. 6.7 around time units 24-28 that show a deep drop



(a) Results for 10% load.

(b) Results for 50% load.



(c) Results for 80% load.

Fig. 6.5: Results of load experiment. (a) The box plot shows the majority of the runs find solutions in under 500 iterations using a 10% system load. The long tails represent the cases where algorithm converges to a local solution and takes time until valid schedule is found. (b) The box plot shows the results converge in under 500 iterations for 50% system loads. (c) The algorithm takes closer to 1,000 solutions to converge for 80% loads. The number of successful runs for 2 processors at 80% load is a result of the task set randomly generated and shows a weakness of the algorithm that relates to the large number of deadlines around $33 \leq t \leq 39$ from Fig. 6.8.

of surplus resources. In such cases, adjusting the start time or deadline for one of T_4. T_6, T_7, T_9, T_{18} or T_{19} would alleviate the problem and make it easier to allocate the tasks. For comparison, the tasks defined in Fig. 6.8 for the $C_N = 4$ with load 80% case did not have

Table 6.4: Parameters for scheduler benchmark experiment.

C_N	**Load**	**Schedule**	**Runs**	**Iterations (Mdn)**	**Missed**	**Cost**
1	10%	Valid	50/50	234.5 ± 178.8 (172.5)	0.0 ± 0.0	0.0 ± 0.0
		Invalid	00/50	N/A	N/A	N/A
	50%	Valid	48/50	234.1 ± 298.1 (136.0)	0.0 ± 0.0	47.3 ± 4.7
		Invalid	02/50	$5,000.0 \pm 0.0$ $(5,000.0)$	1.0 ± 0.0	65.0 ± 35.3
	80%	Valid	50/50	110.5 ± 104.8 (85.0)	0.0 ± 0.0	53.3 ± 5.7
		Invalid	00/50	N/A	N/A	N/A
2	10%	Valid	50/50	390.6 ± 382.5 (267.5)	0.0 ± 0.0	10.6 ± 5.7
		Invalid	00/50	N/A	N/A	N/A
	50%	Valid	50/50	143.1 ± 112.1 (122.0)	0.0 ± 0.0	36.3 ± 7.7
		Invalid	00/50	N/A	N/A	N/A
	80%	Valid	23/50	1434.4 ± 2375.9 (409.0)	0.0 ± 0.0	55.3 ± 3.9
		Invalid	27/50	$10,000.0 \pm 0.0$ $(10,000.0)$	1.0 ± 0.2	56.5 ± 9.2
4	10%	Valid	50/50	324.7 ± 252.7 (248.0)	0.0 ± 0.0	13.2 ± 7.5
		Invalid	00/50	N/A	N/A	N/A
	50%	Valid	50/50	242.7 ± 325.8 (138.5)	0.0 ± 0.0	43.1 ± 11.4
		Invalid	00/50	N/A	N/A	N/A
	80%	Valid	47/50	1256.0 ± 1163.0 (833.0)	0.0 ± 0.0	66.0 ± 6.9
		Invalid	03/50	$10,000.0 \pm 0.0$ $(10,000.0)$	1.0 ± 0.0	52.6 ± 1.5

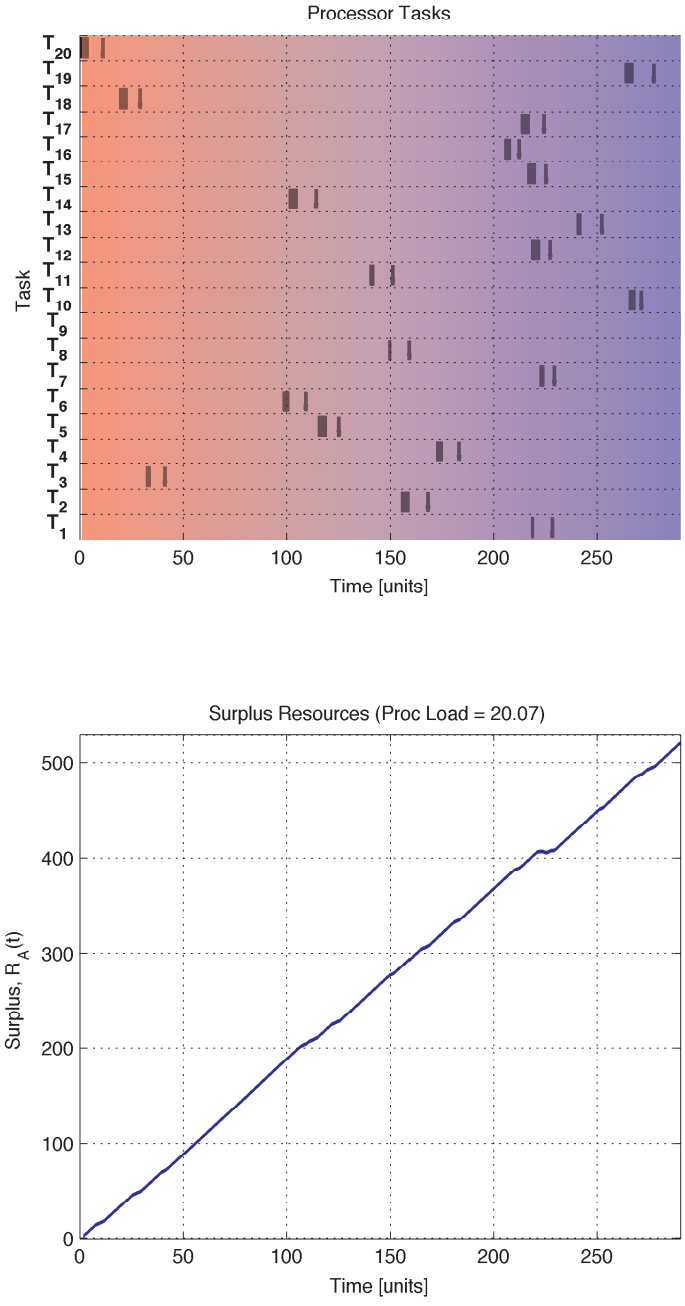


Fig. 6.6: Task definition for $C_N = 2$ with 10% system load (/test06-p2-t20-load10). The processor load estimate assumes a single core.

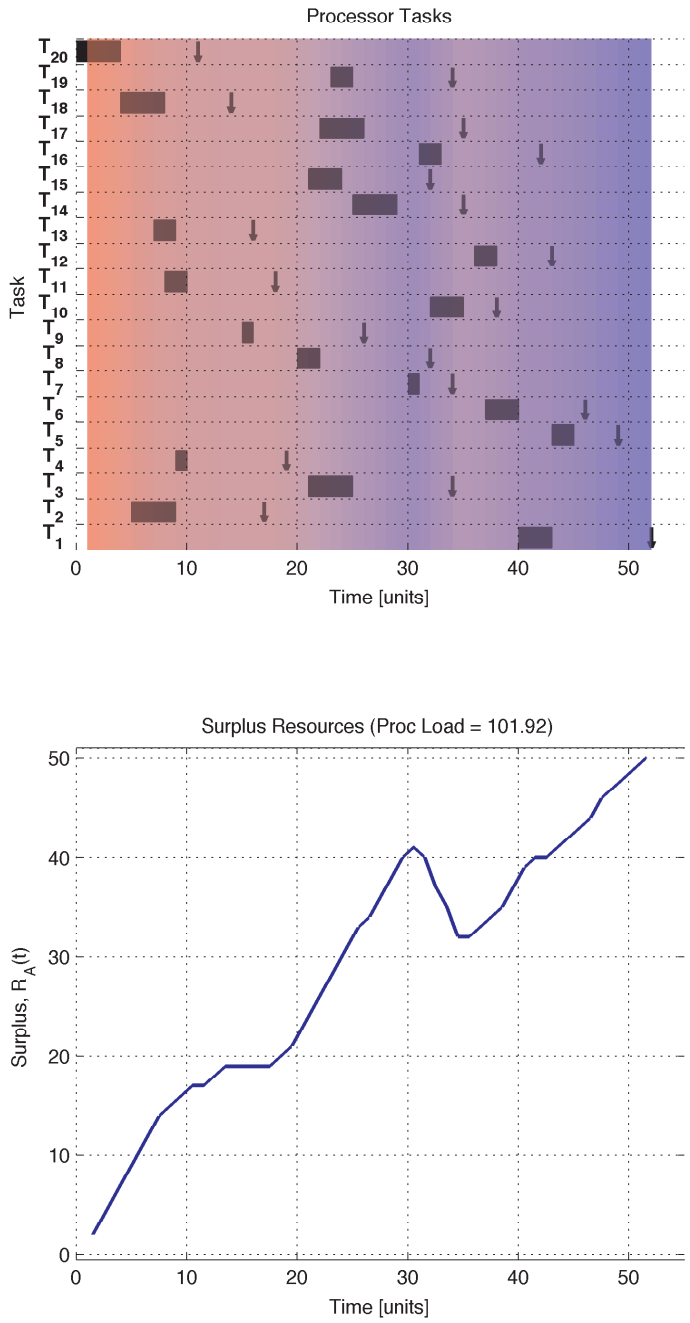


Fig. 6.7: Task definition for $C_N = 2$ with 50% system load (/test07-p2-t20-load50). The processor load estimate assumes a single core.

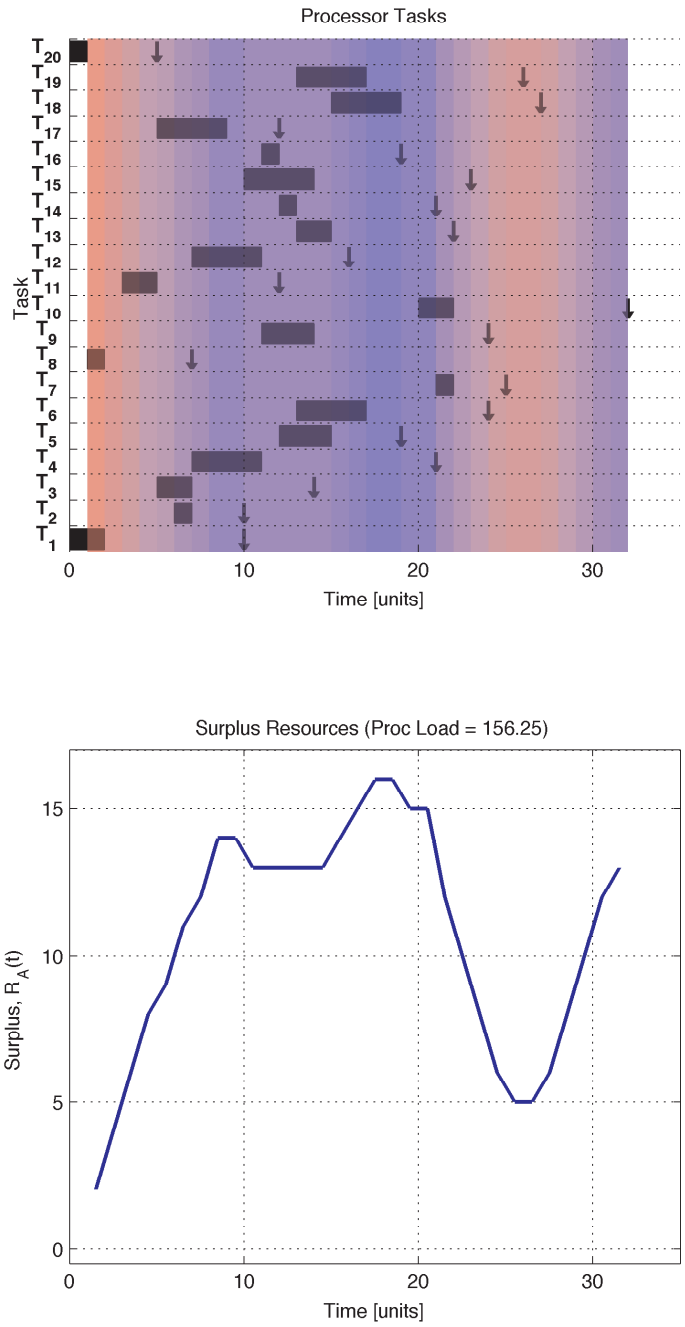
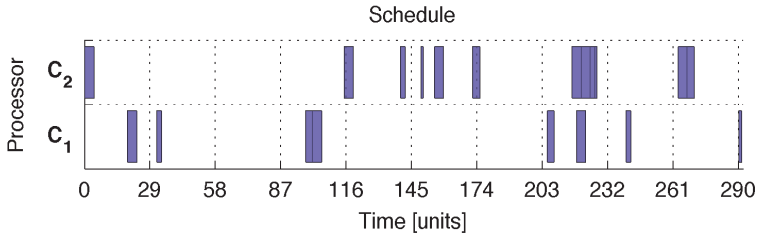


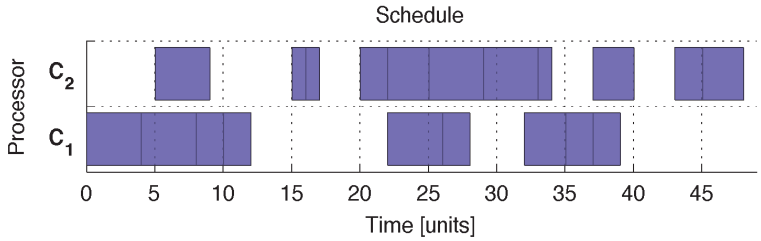
Fig. 6.8: Task definition for $C_N = 2$ with 80% system load (/test08-p2-t20-load80). The processor load estimate assumes a single core.

those features and thus produced much better results. Figures 6.9 show sample schedules generated for $C_N = 2$ under different loads. The difficulties in missed deadlines happen around the expected range of $t \simeq 24$.

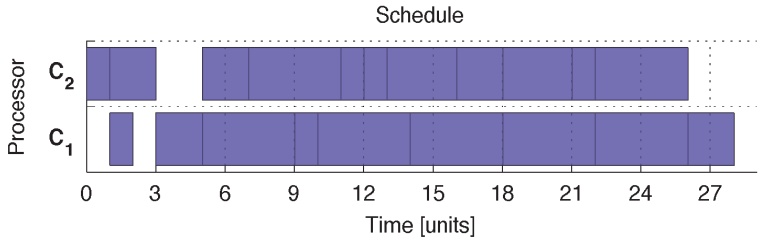
Continuing to exploit the $C_N = 2$ case with 80% load, one can plot the time series for a successful versus an unsuccessful run as shown in Fig. 6.10. For the unsuccessful test (run 008), one can identify an approximate threshold between the transient and the steady state portions of the trajectory at $n = 300$. Using this information, the same analysis from Sec. 4.1.5 can be performed to learn about these components of the trajectory as they define the behaviour for the particle. The moments shown in Fig. 6.11(a) show a WSS signal for the transient and several peaks in the variance for the steady state that could be classified heteroskedastic as there are significant peaks of increasing magnitude for the variance of the signal. This is consistent with the expected behaviours from the test functions used in the thesis. Finally, as shown by the auto-correlation and power spectrum in Fig. 6.12 and 6.13 respectively, the schedule does not show any long term dependance in the trajectories of the steady state portion of the trajectory. This means that once entering that mode, the algorithm begins a random search for better solutions to help escape the trap that is causing some tasks to miss their deadlines. Furthermore, the correlation during the transient is very subtle for this dimension, however depending on the initial location of the tasks relative to its optimal position in the schedule, the correlation can show more significant trends.



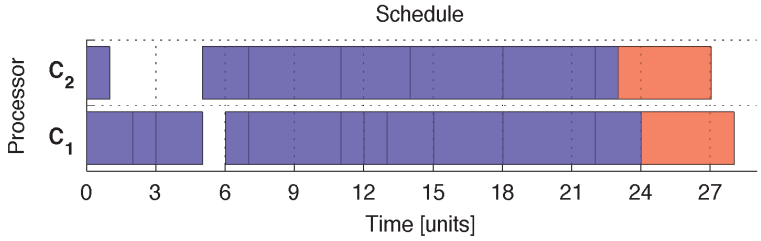
(a) Valid schedule for $C_N = 2$ with 10% load (test06-p2-t20-load10/007/p1.txt).



(b) Valid schedule for $C_N = 2$ with 50% load (test07-p2-t20-load50/017/p18.txt).

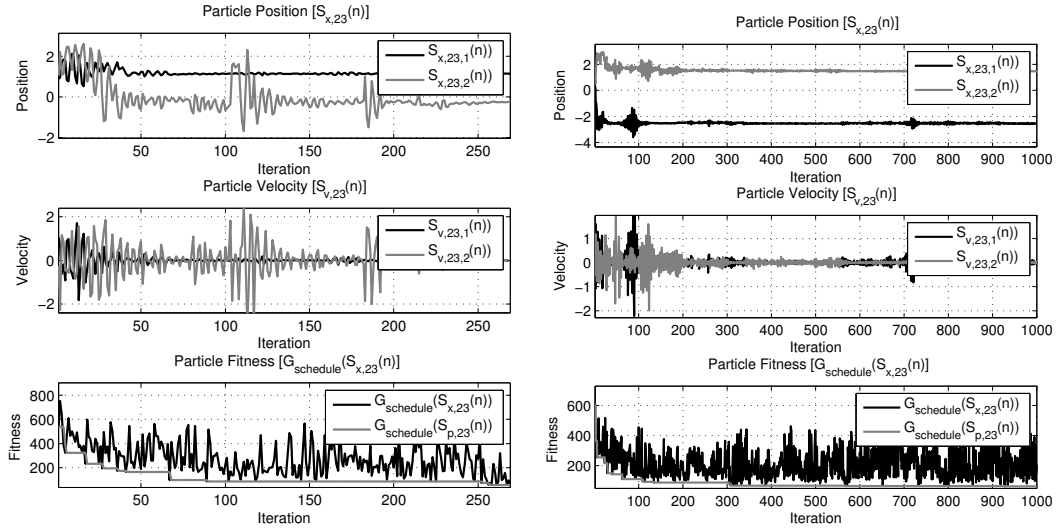


(c) Valid schedule for $C_N = 2$ with 80% load (test08-p2-t20-load80/005/p23.txt).



(d) Invalid schedule for $C_N = 2$ with 80% load (test08-p2-t20-load80/008/p23.txt).

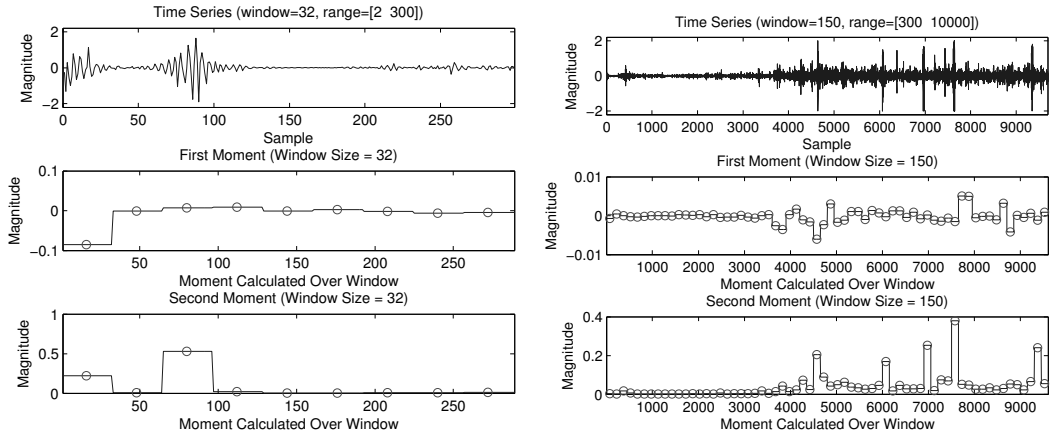
Fig. 6.9: Sample schedules for $C_N = 2$ load experiment. (a) Valid schedule for load of 10%. (b) Valid schedule for load of 50%. (c) Valid schedule for load of 80%. (d) Invalid schedule for load of 80%. Tasks not labeled for legibility.



(a) Trajectory for $S_k = 23$.

(b) Trajectory for $S_k = 23$.

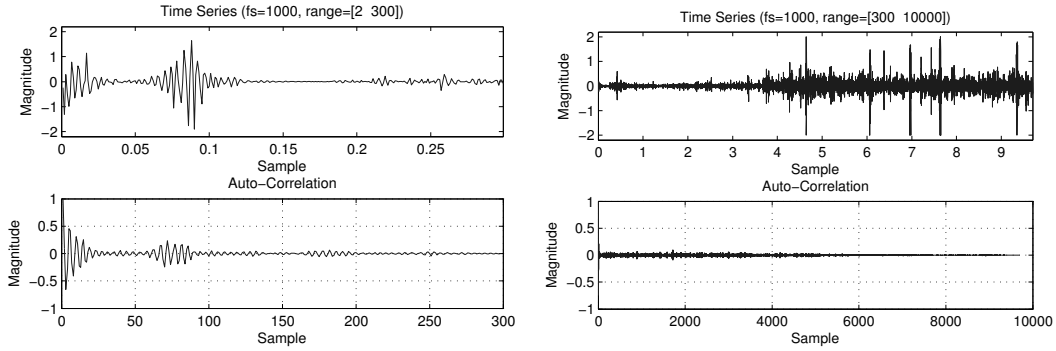
Fig. 6.10: Sample trajectories from $C_N = 2$ with 80% system load experiment. Both graphs show the $d = 1$ and $d = 2$ selected arbitrarily. (a) Results for successful run of the algorithm (test08-p2-t20-load80/005/p23.txt). (b) Results from unsuccessful run of the algorithm (test08-p2-t20-load80/008/p23.txt).



(a) Transient for trajectory.

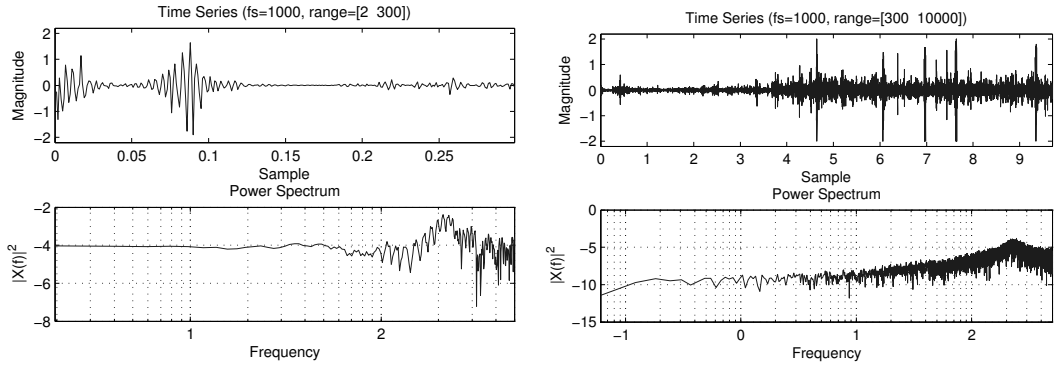
(b) Steady state for trajectory.

Fig. 6.11: First and second moment of particle trajectory for schedule run using $C_N = 2$ with an 80% system load. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory. (test08-p2-t20-load80/008/p23.txt)



(a) Transient for trajectory. (b) Steady state for trajectory.

Fig. 6.12: Auto-correlation of particle trajectory for schedule run using $C_N = 2$ with an 80% system load. (a) Auto-correlation calculated for transient portion of the trajectory. (b) Auto-correlation calculated for the steady state portion of the trajectory. (test08-p2-t20-load80/008/p23.txt)



(a) Transient for trajectory. (b) Steady state for trajectory.

Fig. 6.13: Power spectrum of trajectory for schedule run using $C_N = 2$ with an 80% system load. (a) Power spectrum of trajectory calculated for transient portion of the trajectory. (b) Power spectrum of trajectory calculated for the steady state portion of the trajectory. (test08-p2-t20-load80/008/p23.txt)

6.4 Scalability Experiment

The final experiment explores the performance as the number of tasks, T_N increases keeping all other parameters constant and using $C_N = 4$ processors in the load experiment. The objective is to see the effect of the increased dimensionality resulting from the RK encoding that associates each task to a dimension in the scheduling problem.

The results for the scalability experiment are summarized in Fig. 6.14 and Table 6.5. As expected, the performance deteriorates as the number of tasks increases. The number of iterations for 40 and 50 tasks is similar to the results obtained for a hybrid PSO provided in [CaEC07] with both gBest and ℓBest topologies. Although the performance decreases, the transient portion is still very quick for the 50 task example as shown in Fig. 6.15. Therefore, future improvements to the algorithm must focus on the performance in the steady state portion of the trajectory.

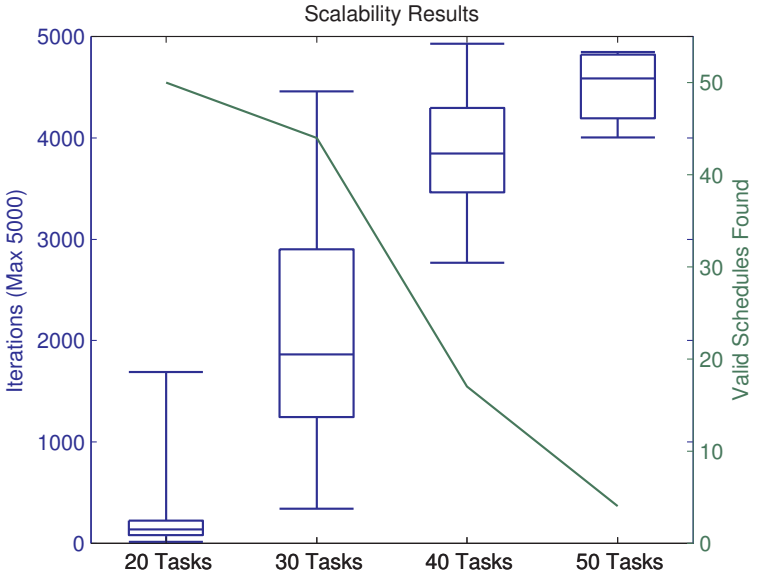


Fig. 6.14: Results of scalability experiment. The box plot shows how the algorithm takes longer to find solutions as the number of tasks increases, while the number of valid schedules found in the maximum 5,000 iterations decreases as the problem size increases.

Table 6.5: Parameters for scheduler benchmark experiment.

T_N	Schedule	Runs	Iterations (Mdn)	Missed	Cost
20	Valid	50/50	242.7 ± 325.8 (138.5)	0.0 ± 0.0	43.1 ± 11.4
	Invalid	00/50	N/A	N/A	N/A
30	Valid	44/50	2136.5 ± 1170.3 (1864.0)	0.0 ± 0.0	48.5 ± 15.0
	Invalid	06/50	$5,000.0 \pm 0.0$ $(5,000.0)$	1.0 ± 0.0	32.0 ± 6.3
40	Valid	17/50	3872.4 ± 597.5 (3849.0)	0.0 ± 0.0	60.2 ± 13.3
	Invalid	33/50	$5,000.0 \pm 0.0$ $(5,000.0)$	1.3 ± 0.6	65.0 ± 20.1
50	Valid	04/50	4508.2 ± 394.0 (4587.0)	0.0 ± 0.0	83.0 ± 14.4
	Invalid	46/50	$5,000.0 \pm 0.0$ $(5,000.0)$	3.2 ± 1.7	109.2 ± 35.3

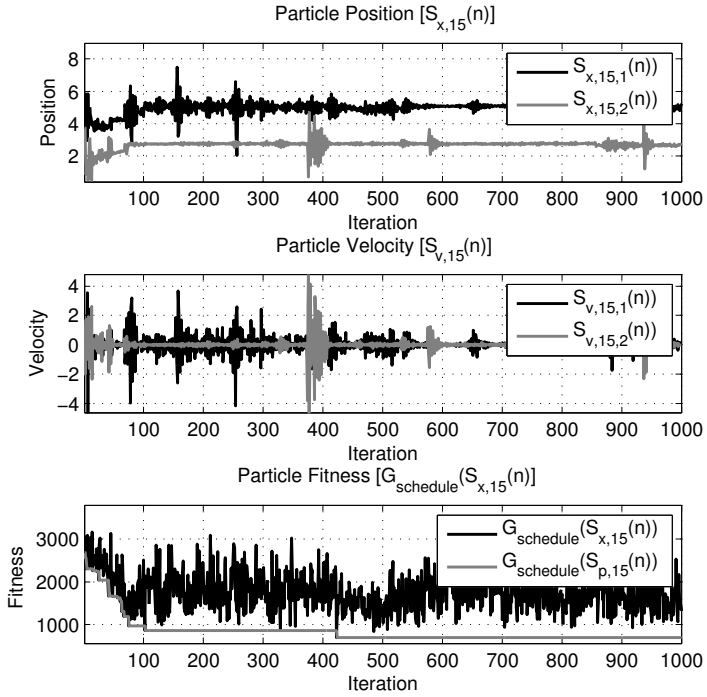


Fig. 6.15: Sample trajectories from $C_N = 4$ with 50% scalability experiment with 50 tasks. Both graphs show the $d = 1$ and $d = 2$ selected arbitrarily. (test08-p2-t20-load80/005/p23.txt).

6.5 Summary

This chapter described the three experiments used to test the performance of the scheduler for extreme examples, different system loads, and varying numbers of tasks. The trajectory of the particles in the scheduler presented (i) confirm the behaviours extracted in Sec. 4.1.4 still apply, (ii) demonstrate the effectiveness of PSO during the transient phase, and (iii) show the weaknesses of the algorithm when performing local searches during the steady-state phase. The following chapter summarizes the results and contributions from this study as they pertain to both scheduling and the study of PSO trajectories.

Chapter 7

Conclusions

7.1 Overview

This thesis presents a scheduler algorithm for symmetric multiprocessors (SMP) for real-time applications, such as small satellites. The scheduler uses the particle swarm optimization (PSO) algorithm to find valid solutions that meet the requirements for the system as described in Ch. 4. This is supported through a data-driven study of the particle trajectories that aid in the selection of parameters and evaluation of both the optimization and scheduler algorithm performance. In Ch. 6, three main experiments are used to demonstrate the capabilities of the scheduler. The experiments include (i) a benchmark set of problems with zero slack in all tasks, (ii) using different system loads, and (iii) finally testing on different problem sizes. The results indicate the algorithm can find solutions for small task sets ($T_N \leq 30$). The data driven analysis of trajectories for the schedule reveals that the algorithm can find partial solutions in very few iterations proportional to the problem size, but fails to escape these local minima to produce a valid schedule. This is consistent with the behaviour observed for other test functions where the directed movement of the particles

fades upon reaching a local solution.

7.2 Thesis Conclusions

This thesis attempted to answer many interesting research questions about evolutionary algorithms (EAs), PSO, and scheduling algorithms, as outlined in Sec. 1.2.3. This section links back the results and observations to the research questions to provide insight into the answers and potential future research.

The long-term dependencies in the behaviour of particles serve as a significant indicator of the performance of EAs, as these indicate whether the system is evolving towards a solution, or whether it has reached its limitations and is proceeding with small random perturbations. This is reinforced through the power spectrum analysis that evaluates the transient and steady state portions of particle trajectories independently. The negative slope in the spectrum of the transient shows the explicit directed movement of individual particlesThe particle ensemble analysis indicates that the topology has a larger influence on the trajectories than the personal and social weights. This was reinforced by fixing the initial conditions for particles and showing the weights gave rise to heteroskedastic behaviours in the trajectories, but did not affect the overall trajectory of the particles.

For this thesis, a typical particle in the swarm is selected for analysis based on the mean square error (MSE) of individual trajectories with respect to an ensemble mean. The ensemble constructed from repeated runs with fixed initial conditions. The typical particle showed a transient and steady state phases, with corresponding properties, that were consistent across different cost functions. Although the typical particle reveals many interesting behaviours and characteristics of the movement of the swarm, further research into some other cases, such as the worst behaving particle with greatest MSE, are required to fully comprehend the behaviour of the swarm.

In an effort to produce results that could be evaluated against previously published material, the convergence rate was used as an indicator into the performance of the PSO algorithm. Although this is effective, as shown in Tables 6.4 and 6.5, the mean behaviours may be misleading indicators of the performance of the algorithm that do not tell the full story. The distribution of solutions shows what may be described as outliers from particle trajectories that spent many iterations testing semi-random solutions during the steady-state portion of the trajectory. This is more pronounced in the thesis because there is a very well defined termination criterion in terms of missed deadlines. However, in continuous problems (i.e., not sequencing in scheduling), such criteria are difficult to define because the small variations can appear to always improve the fitness much more so than in the discretized set of sequences obtained through the RK encoding for scheduling. For more implementation independent measures, it is necessary to find a more clear means of identifying the transition between the transient and steady state portions of the algorithm, as that iteration number would be more indicative of the performance than using the final count.

The framework described in Sec. 4.1.4 can be used to determine the impact of different PSO parameters, such as topologies and weights, on the trajectories of the particles. This gives insight into the different levels of exploration that can be achieved through changes in topologies as shown in [ScKA10], since the analysis is implementation independent, thus it can be used to complement the convergence rate metrics commonly used for comparing EAs. A full sensitivity analysis is not provided as part of the thesis. The techniques presented can lead to a better understanding for particular problems.

Finally, the scheduler implemented demonstrates that EAs can be used for scheduling in SMP. A cost function based on minimum total tardiness (MTT) with cumulative penalties is presented in Sec. 4.2.3 and tested extensively in Sec. 4.2.5. The results indicate the algorithm can find solutions consistently for up to 30 tasks under different system loads,

however, it does not scale very well because of the random keys (RK) encoding used. The proposed scheduler is designed around cumulative penalties such that it can be adjusted for different scenarios including task precedence, priorities, and processor affinity.

7.3 Contributions

This thesis contributes to new knowledge to the performance and behaviour of PSO as well as novel implementations for scheduling in SMP systems. These contributions are outlined below.

(a) A novel data-driven analysis of PSO particle trajectories is presented and consists of looking at the step size (or velocity) of the particles along single dimensions to extract characteristics in the behaviour. In particular, the analysis revealed a clear distinction in the directed movement of the particles during the transient phase, which showed the long-term trends in the behaviour, as compared to the steady-state portions, where perturbations appear to be more random as the particle explores the local space. The data-driven analysis revealed the importance of the swarm topology compared to the stochastic processes in the personal and social weights. For a fixed set of initial conditions and a given topology, the particles' movement is well defined with minor variations at the tail end of the trajectories. This is a very important finding that can also be used to evaluate and compare different topologies to understand how the behaviour of the individual particles (microcosm) and the swarm (macrocosm) work under different conditions.

(b) A complexity estimate for PSO is provided with some insights into improvements to the algorithm that were implemented and tested as part of this thesis. Such metrics are often left out in the study of EAs, but are needed in order to estimate and verify whether such algorithms can be used in embedded applications.

(c) A novel cost function for scheduling aperiodic and independent real-time tasks based on MTT and cumulative penalties is presented in this thesis. The algorithm performs well under small problem sets, $T_N \leq 30$, and shows a very fast transient towards good solutions. This demonstrates the evolutionary processes behind the scheduler are working as designed. Since valid schedules are not produced 100% of the time, it is recommended that this be used either for non-critical applications.

(d) Finally, a task generator and simulator are described such that the system load and the amount of surplus resources are both taken into account. This is a novel combination of a schedulability test with task generation and can be used in other applications to model different kinds of behaviours. In this thesis, uniform distributions of tasks were needed, while by changing the trend in the $R_A(t)$ curve, one can generate tasks sets that match those observed in different applications.

7.4 Limitations and Potential Solutions

The limitations in the work presented are divided into two categories: (i) swarm analysis, and (ii) scheduling.

The analysis of PSO demonstrated that there is a clear distinction in the trajectory of the particles as they transition from the transient to the steady-state mode. It is difficult to identify the key point where the transition takes place. In this thesis, the value was selected by inspecting the plots and selecting a value in the appropriate range. Identifying such point is very important because spending extra time searching can be shown to have little to no impact on finding the solution and therefore is detrimental to the process. Therefore, to help identify that range, multi-scale techniques that extract the long-term relationship are needed. One suggestion would be to use the *variance fractal dimension trajectory* (VFDT) as it serves in classification of signals and can detect changes in the

correlation of steps through a sliding window and is not susceptible to possible outliers in the trajectory [Kins09]. Using that information, the search then needs to be modified to expedite the process for the particular application.

In the scheduling algorithm, the main limitations came from the characteristics of the parameter space defined through the RK encoding. In essence, trying to map discrete permutations of the schedule to a continuous space for particles to move through causes many flat areas where particles can get stuck. These plateau regions do not have a slope in either direction and thus make it difficult for particles to determine which direction to move in order to improve the search. In the early stages of the algorithm, this is not a serious issue because the forces from the other particles in the neighbourhood are sufficient to make the particle move large distances and avoid these flat surfaces. However, as the particles converge in one area, the problem becomes more difficult to solve. The approach taken in the thesis used a directed mutation for the dimension associated with a task that missed a deadline. Although this helps the algorithm in some cases and produces some of the high iteration count solutions, the method is not entirely effective. Another approach would involve using an alternative encoding mechanism that retains the benefits of a continuous PSO while smoothing out the parameter space.

Finally, the dimensionality of the scheduler is again limited by the RK encoding. By mapping each task to a unique dimension, the problem becomes very difficult to solve. To overcome this, new encoding techniques are needed that can group information through hash-like functions and provide unique identifiers in a continuous domain.

References

[Ange98] Peter Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," *Evolutionary Programming VII*, vol. 1447, pp. 601–610, 1998.

[BSDK13] Kuran Budhraja, Ashutosh Singh, Gaurav Dubey, and Arun Khosla, "Exploration Enhanced Particle Swarm Optimization using Guided Re-Initialization," *in Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2012* (Gwalior, India; December 14-16, 2012) *in* Jagdish Bansal, Pramod Singh, Kasum Deep, Millie Pant, and Atulya Nagar, editors, *Advances in Intelligent Systems and Computing*, pp. 403–416, Spinger Berlin Heidelberg, 2012. {ISBN 978-81-322-1037-5}

[BeCa06] S. Beaudette, J. de Carufel, D. McCabe, W. Whitehead, R. Buckingham, C. Pye, P. Tremblay, D. Kefallinos, and A. Thompson, "RADARSAT Constellation Mission Phase A Spacecraft Bus Development – Work to Date," *in Proc. of the 13th CASI Aeronautics Conference, ASTRO 2006*, (Montreal, QC, Canada; April 2006), pp. 1–11, 2006.

[BeEn06] Frans van den Bergh and Andries Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[Bean94] James C. Bean, "Genetics and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[Berg01] Frans van den Bergh, *An Analysis of Particle Swarm Optimizers*, Ph.D. thesis, University of Pretoria, Pretoria, South Africa, 300 pp., 2001.

[Boyd04] Stephen Boyd and Lieven Vandenberghe, *Convex Optimization*, Cambridge University Press, New York, NY, USA, 2004, 727 pp. {ISBN 978-0521833783}

[BrKe07] Daniel Bratton and James Kennedy, "Defining a Standard for Particle Swarm Optimization," *in Swarm Intelligence Symposium, 2007, SIS 2007*, (Honolulu, Hawaii, USA; April 1-5, 2007), pp. 120–127, 2007.

[Bruc95] Peter Brucker, *Scheduling Algorithms*, Springer-Verlag, Secacus, NJ, USA, 1995, 326 pp. {ISBN 978-3540600879}

[CaEC07] Leticia Cagnina, Susana Esquivel, and Carlos Coello, "Hybrid Particle Swarm Optimizers in the Single Machine Scheduling Problem: An Experimental Study," *in* Keshav Dahal, Kay Chen Tan, and Peter Cowling, editors, *Evolutionary Scheduling*, vol. 49 of *Studies in Computational Intelligence*, pp. 143–164, Springer Berlin Heidelberg, 2007. {ISBN 978-3-540-48582-7}

[Chen02] Albert Cheng, *Real-Time Systems Scheduling, Analysis and Verification*, John Wiley & Sons, Inc., New York, NY, USA, 2002, 552 pp. {ISBN 978-0471184065}

[ChRJ06] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen, "An Optimal Real-Time Scheduling Algorithm for Multiprocessors", *in Proc. of the 27th IEEE International Real-Time Systems Symposium, RTSS06*, (Rio de Janeiro, Brazil; December 5-8, 2006), pp. 101–110, 2006.

[ClKe02] Maurice Clerc and James Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[CoMM67] Richard Conway, William Maxwell, and Louis Miller, *Theory of Scheduling*, Addison-Wesley Publishing Company, 1967, 294 pp. {ISBN 978-0486428178}

[Cook10] Darcy Cook, *A Multiprocessing System-on-Chip Framework Targeting Stream-Oriented Applications*, Master's thesis, University of Manitoba, Winnipeg, Canada, 189pp. 2010.

[CoFe13] Darcy Cook and Ken Ferens, "Performance Analysis of a Reconfigurable Shared Memory Multiprocessor System for Embedded Applications," *Journal of ICT Research and Applications*, February 2013.

[DeMo89] Michael Dertouzos and Aloysius Ka-Lau Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.

[DePW04] A. Denis, S. Page, and I. Walkty, "The Evolution of a SCISAT-1 Spacecraft to Provide a Generic Small Satellite Bus for the Canadian Space Agency," *in Proc. of the 55th Int. Astronautical Congress, IAC 2004*, (Vancouver, BC, Canada; October 4-8, 2004), vol. 5, pp. 3362–3370, 2004.

[DoMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on System, Man, and Cybernetics – Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[Dvor09] Daniel Dvorak, "NASA Study on Flight Software Complexity," Tech. rep., Systems and Software Division, Jet Propulsions Laboratory, NASA, 2009.

[EbKe95] Russell Eberhart and James Kennedy, "A new optimizer using particle swarm theory," *in Proc. of the Sixth International Symposium on Micro Machine and Human Science, MHS 1995*, (Nagoya, Japan; October 4-6, 1995), pp. 39–43, 1995.

[EbSh00] Russell Eberhart and Yuhui Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," *in Proc. of the 2000 Congress on Evolutionary Computation, CEC 2000*, (La Jolla, California, USA; July 16-19, 2000), vol.1, pp. 84–88, 2000.

[EbSh07] Russell Eberhart and Yuhui Shi, *Computational Intelligence Concepts to Implementations*, Morgan Kaufmann Publishers, Burlington, MA, USA, 2007, 467 pp. {ISBN 978-1558607590}

[FlMa08] Dario Floreano and Claudio Mattiussi, *Bio-Inspired Artificial Intelligence*, MIT Press, Cambridge, MA, USA, 2008, 659pp. {ISBN 978-0-262-06271-8}

[FoBe95] David Fogel and Hans-Georg Beyer, "A note on the empirical evaluation of intermediate recombination," *Evolutionary Computing*, vol. 3, no. 4, pp. 491–495, 1995.

[Fogo97] David Fogel, "The Advantages of Evolutionary Computation," *in Proc. of Biocomputing and Emergent Computation, BCEC1997*, (Skvde, Sweden; September 1-2, 1997), 11 pp., 1997.

[Forr91] Stephanie Forrest, *Emergent Computation*, MIT Press, Cambridge, MA, USA, 1991, 452pp. {0-262-56057-7}

[FuNa10] Shelby Funk and Vijaykant Nadadur, "LRE-TL: An Optimal Multiprocessor Scheduling Algorithm for Sporadic Task Sets", *ACM Journal of Real-Time Systems*, vol. 46, no. 3, pp. 332–359, 2010.

[GOLD09] –, "Rules for the Design, Development, Verification, and Operation of Flight Systems," Goddard Technical Standard GSFC-STD-1000E, NASA, Goddard Space Flight Center, Greenbelt, MD 20771, 2009.

[Glov89] Fred Glover, "Tabu Search - Part 1," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[Glov90] Fred Glover, "Tabu Search - Part 2," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

[GrKi98] J. Greenberg and W. Kinsner, "Characterizing behaviour of birth/death processes using fractal measures," *in Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 1998*, (Waterloo, ON, Canada; May 24-28, 1998) pp. 373–376, 1998.

[Gree11] William Greene, *Econometric Analysis*, Prentice Hall, 7th edn., 2011, 1232 pp. {ISBN 978-0131395381}.

[HaKo10] Raymond Harris, Diane Kotelko, Ian Walkty, Walter Czyrnyj, Paul Harrison, and Dave McCabe, "Small Satellite Bus Evolution at Magellan Aerospace," *in Proc. of the 15th CASI Aeronautics Conference, ASTRO 2010*, (Montreal, QC, Canada; May 4-6, 2010), pp. 1–14, 2010.

[HaLo05] David Harland and Ralph Lorenz, *Space Systems Failures - Disasters and Rescues of Satellites, Rockets, and Space Probes*, Springer Praxis Books, 2005, 368 pp. {ISBN 978-0387215198}

[HaMB06] P. Harrison, D. McCabe, S. Beaudette, P. Gregory, and A. Scott, "Accommodating the Chinook Mission Pointing Requirements with the Magellan MAC-200 SmallSAT Bus," *in Proc. of the 13th CASI Aeronautics Conference, ASTRO 2006*, (Montreal, QC, Canada; April 2006), pp. 1–15, 2006.

[Holl75] John H. Holland, *Adaptation in Natural and Artificial Systems - An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, 1975, 183 pp. {ISBN 978-0472084609}

[John01] Steven Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*, Scribner, 2001, 288 pp. {ISBN 978-0684868769}

[Jong75] Kenneth Alan De Jong, *An analysis of the behavior of a class of genetic adaptive systems.*, Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA, 268pp., 1975.

[KSFC12] Witold Kinsner, Dario Schor, Mohammadreza Fazel-Darbandi, Brendan Cade, Kane Anderson, Cody Friesen, Diane Kotelko, and Philip Ferguson, "The T-Sat1 Nanosatellite Team of Teams," *in Proc. of the 11th IEEE International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2012*, (Kyoto, Japan; August 22-24, 2012), pp. 1–11, 2012.

[KZHZ11] Ayda Kaddoussi, Nesrine Zoghlami, Slim Hammadi, and Hayfa Zgaya, "An Agent-based distributed scheduling for military logistics," *in Proceedings of the 11th International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 134 –140, 2011.

[KaCa03] Tom Kalisker and Doug Camens, "Solving Mastermind Using Genetic Algorithms," *in Proc. of the Genetic and Evolutionary Computation Conference*, (Chicago, IL, USA; July 12-16, 2003), pp. 1590–1591, 2003.

[KaSc04] Holger Kantz and Thomas Schreiber, *Nonlinear Time Series Analysis*, Cambridge University Press, 2nd edn., 2004, 388pp. {ISBN 978-0521529020}

[KeEb01] James Kennedy and Russell Eberhart, *Swarm Intelligence*, Morgan Kaufmann, 2001, 512pp. {ISBN 978-1558605954}

[KeEb95] James Kennedy and Russell Eberhart, "Particle swarm optimization," *in Proc. of the IEEE International Conference on Neural Networks, 1995*, (Perth, WA, USA; November 27-December 1, 1995), vol. 4, pp. 1942–1948, 1995.

[KeEb97] James Kennedy and Russell Eberhart, "A discrete binary version of the particle swarm algorithm," *in Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, (Orlando, FL, USA; October 12-15, 1997), vol. 5, pp. 4104–4108, 1997.

[KeMe02] James Kennedy and Rui Mendes, "Population structure and particle swarm performance," *in Proc. of the 2002 Congress on Evolutionary Computing, CEC 2002*, (Honolulu, Hawaii, USA; May 12-17, 2002), pp. 1671–1676, 2002.

[Kenn07] James Kennedy, "Some Issues and Practices for Particle Swarms," *in IEEE Swarm Intelligence Symposium, SIS 2007*, (Honolulu, Hawaii, USA; April 1-5, 2007), pp. 162–169, 2007.

[Kenn97] James Kennedy, "The particle swarm: social adaptation of knowledge," *in Proc. of the IEEE International Conference on Evolutionary Computation, CEC 1997*, (Indianapolis, IN, USA; April 13-16, 1997), pp. 303–308, 1997.

[Kenn99a] James Kennedy, "Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance," *in Proc. of the 1999 Congress on Evolutionary Computation, CEC 1999*, (Washington, DC, USA; July 6-9, 1999), pp. 1931–1938, 1999.

[Kins04] Witold Kinsner, "Is entropy suitable to characterize data and signals for cognitive informatics?" *in Proc. of the Third IEEE International Conference on Cognitive Informatics, ICCI 2004*, (Victoria, BC, Canada; August 16-17, 2004), pp. 6–21, 2004.

[Kins05] Witold Kinsner, "Some advances in cognitive informatics," *in Proc. of the Fourth IEEE International Conference on Cognitive Informatics, ICCI 2005*, (Irvine, CA, USA; August 8-10, 2005), pp. 6–7, 2005.

[Kins05b] Witold Kinsner, "A unified approach to fractal dimensions," *in Proc. of the Fourth IEEE International Conference on Cognitive Informatics, ICCI 2005*, (Irvine, CA, USA; August 8-10, 2005), pp. 58–72, 2005.

[Kins09] Witold Kinsner, "Fractal and Chaos Engineering," Course Notes; Winnipeg MB; University of Manitoba, 2009, 900pp.

[Kins12] Witold Kinsner, "Microprocessing Interfacing for Real-Time Systems Course Notes," Tech. rep., Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 2012.

[Kirk83] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

[Koen85] Billy Vaughn Koen, *Definiition of the Engineering Method*, American Society for Engineering, 1985, 74 pp. {ISBN 978-0878231010}

[LeRB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of Machine Scheduling Problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[Leho96] J. Lehoczky, "Real-time queueing theory," *in Proc. of the IEEE Real-Time Systems Symposium*, (Los Alamitos, CA, USA; December 4-6, 1996), pp. 186–195, 1996.

[LiWJ07] Bo Liu, Ling Wang, and Yi-Hui Jin, "An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 37, no. 1, pp. 18–27, 2007.

[Liu00] Jane W. S. Liu, *Real-time systems*, Prentice Hall, New Jersey, USA, 2000, 592 pp. {ISBN 978-0130996510}

[LiuW05] Bo Liu, Ling Wang, Yi-Hui Jin, Fang Tang, and De-Xian Huang, "Improved particle swarm optimization combined with chaos," *Chaos, Solitons & Fractals*, vol. 25, no. 5, pp. 1261–1271, 2005.

[MAW09] Neil Gadhok, Dario Schor, Diane Kotelko, Ian Walkty, and Walter Czyrnyj, "ER103105A RCM Communications and Data Processing Analysis (SDRL EN43-B)." Tech. rep., Magellan Aerospace Winnipeg, 2009.

[Math00] Mohit Mathur, Sachin Karale, Sidhartha Priye, V. K. Jayaraman, and B. D. Kulkarni, "Ant Colony Approach to Continuous Function Optimization," *Industrial & Engineering Chemistry Research*, vol. 39, no. 10, pp. 3814–3822, 2000.

[MeKN04] Rui Mendes, James Kennedy, and Jose Neves, "The fully informed particle swarm: simpler, maybe better," *IEEE Transactions on Evolutionary Computing*, vol. 8, no. 3, pp. 204–210, 2004.

[Mend04] Rui Mendes, *Population Topologies and Their Infuence in Particle Swarm Performance*, Ph.D. thesis, Escola de Engenharia, Universidade do Minho, Braga, Portugal, 184pp., 2004.

[Nell11] Peter Nell, "Reconfigurable Processing Platform - FPGA Based, Multi-core, Dynamically Reconfigurable Processing Platform," *in Proceedings of the Canadian Space Agency Workshop on the Utilization of Field Programmable Gate Arrays (FPGA's) in Canadian Space Missions*, p. 14, Magellan Aerospace, 2011.

[OzMo99] Ender Ozcan and Chilukuri Mohan, "Particle swarm optimization: surfing the waves," *in Proc. of the 1999 Congress on Evolutionary Computation, CEC 1999*, (Washington, DC, USA; July 6-9, 1999), vol. 3, pp. 1939–1944, 1999.

[PaHa06] Steve Page, Raymon Harris, Ian Walkty, Dave Beattie, and Harold Dahl, "The Development of the MAC-200 Small Satellite Bus The Development of the MAC-200 Small Satellite Bus for the Canadian Space Agency," *in Proc. of the 13th CASI Aeronautics Conference, ASTRO 2006*, (Montreal, QC, Canada; April 2006), pp. 1–12, 2006.

[PeBr04] Walter Peruzzini and Gilles Brassard, "Canadian Multi-Mission Small and Micro Satellite Buses," *in Proc. of the 55th Int. Astronautical Congress, IAC 2004*, (Vancouver, BC, Canada; October 4-8, 2004), vol. 6, pp. 3632–3635, 2004.

[PrZe09] Freddy Pranajaya and Robert Zee, "The Generic Nanosatellite Bus: From Space Astronomy to Formation Flying Demo to Responsive Space," *in Proc. of the First International Conference on Advances in Satellite and Space Communications, SPACOMM 2009*, (Colmar, France; July 20-25, 2009), pp. 134 –140, 2009.

[RaSt94] Krithi Ramamritham and John Stankovic, "Scheduling algorithms and operating systems support for real-time systems," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 55–67, 1994.

[RiVe02] Jacques Riget and Jakob Vesterstoem, "A diversity-guided particle swarms optimizer - the ARPSO," Tech. Rep. 2002-02, Dept. of Computer Science, University of Aarhus, Denmark, 13 pp., 2002.

[SSNK09] Dario Schor, Jane Polak Scowcroft, Christopher Nichols, and Witold Kinsner, "A command and data handling unit for pico-satellite missions," *in Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2009*, (St. John's, NL, Canada; May 3–6, 2009), pp. 868–873, 2009..

[ScKA10] Dario Schor, Witold Kinsner, and John Anderson, "A Study of Optimal Topologies in Swarm Intelligence," *in IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2010*, (Calgary, AB; May 2–5, 2010), pp. 1–8, 2010.

[ScKM11] Dario Schor, Witold Kinsner, and Kathryn Marcynuk, "Reinforcing the design foundation of asynchronous serial data communications using logic and protocol analyzers," *in Proc. of the Canadian Engineering Education Association Conference, CEEA 2011*, (St. John's, NL, Canada; June 6–8, 2011), pp. 1–6, 2011.

[ScKi10] Dario Schor and Witold Kinsner, "A Study of Particle Swarm Optimization for Cognitive Machines," *in Proc. of the 9th IEEE International Conference on Cognitive Informatics, ICCI 2010*, (Beijin, China; July 7–9, 2010), pp. 26–33, 2010.

[ScKi11a] Dario Schor and Witold Kinsner, "Time and Frequency Analysis of Particle Swarm Trajectories for Cognitive Machines," *International Journal of Cognitive Informatics and Natural Intelligence*, vol. 5, no. 1, pp. 18–41, 2011.

[ScKi11b] Dario Schor and Witold Kinsner, "Towards Agent Swarm Optimization," *in Proc. of the 10th IEEE International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2011*, (Banff, AB, Canada; August 18–20, 2011), pp. 227–234, 2011.

[SeMa09] Davoud Sedighizadeh and Ellips Masehian, "Particle Swarm Optimization: Methods, Taxonomy, and Applications," *International Journal of Computer Theory and Engineering*, vol. 1, no. 5, pp. 486–502, 2009.

[ShEb98] Yuhui Shi and Russell Eberhart, "A modified particle swarm optimizer," *in Proc. of the IEEE International Conference on Evolutionary Computation*, (Anchorage, AK, USA; May 4-9, 1998), pp. 69–73, 1998.

[ShEb99] Yuhui Shi and Russell Eberhart, "Empirical study of particle swarm optimization," *in Proc. of the 1999 Congress on Evolutionary Computation, CEC 1999*, (Washington, DC, USA; July 6-9, 1999), vol. 3, pp. 1945–1950, 1999.

[ShKi96] D. Shaw and Witold Kinsner, "Chaotic simulated annealing in multilayer feedforward networks," *in Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 1996*, (Calgary, AB, Canada; May 26-29, 1996), vol. 1, pp. 265 –269, 1996.

[SoDo08] Krzysztof Socha and Marco Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, pp. 1155–1173, 2008.

[Soch08] Krzysztof Socha, *Ant Colony Optimization for Continuous and Mixed-Variable Domains*, Ph.D. thesis, IRIDIA, CoDE, University Libre de Bruxelles, Brussels, Belgium, 193 pp., 2008.

[SpYo89] Timothy Spinney and Hassan Yousef, "Standard satellite data bus initiative," *in Proc. of the IEEE Aerospace Applications Conference*, (Breckenridge, CO, USA; February 12-17, 1989), pp. 10, 1989.

[TBLW11] Terry Tidwell, Carter Bass, Eli Lasker, Micah Wylde, Christopher Gill, and William Smart, "Scalable Utility Aware Scheduling Heuristics for Real-time Tasks with Stochastic Non-preemptive Execution Intervals," *in Proc. of the 23rd Euromicro Conference on Real-Time Systems, ECRTS 2011*, (Porto, Portugal; July 5-8, 2011), pp. 238–247, 2011.

[Trel03] Ioan Cristian Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317–325, 2003.

[WaSt99] Duncan Watts and Steven Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[WeBe06] V. A. Wehrle, S. Beaudette, D. McCabe, P. Harrison, and I. R. Mann, "ORBITALS SmallSAT Mission Overview and Spacecraft Concept," *Proc. of the 13th CASI Aeronautics Conference, ASTRO 2006*, (Montreal, QC, Canada; April 2006), pp. 1–13, 2006.

[WeLa99] James Wertz and Wiley Larson, *Space Mission Analysis and Design*, Microcosm Press, 3rd edn., 1999. 976 pp. {ISBN 978-1881883104}

[WoMa97] David Wolpert and William Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[ZiPL05] Karin Zielinski, Dagmar Peters, and Rainer Laur, "Stopping Criteria for Single-Objective Optimization," *in Proc. of the Third Int. Conference on Computational Intelligence, Robotics and Autonomous Systems, CIRAS 2005*, (Singapore; December 14-16, 2005), 6 pp., 2005.

There are 92 references included in this thesis. They range from 1967 through 2013, with the majority from the early 2000s.

Appendix A

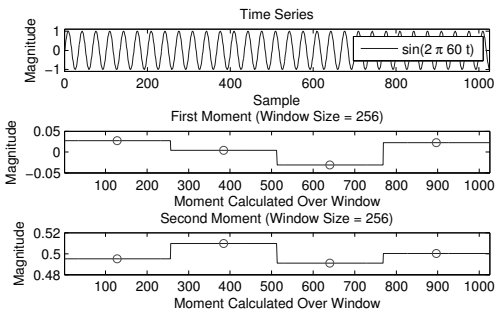
Particle Trajectory Analysis

This appendix contains a collection of the graphs studied as part of the trajectory study described in Sec. 4.1.4. Section A.1 contains the results of the verification tests for moments, correlation, and power spectrum tests. Following that, Sec. A.2-A.5 contain the plots analyzed for the Sphere, Rosenbrock, Rastrigin, and Griewank functions respectively. Some of the more significant results appeared in the main body of the text and are repeated here as a means of providing a complete and compact set.

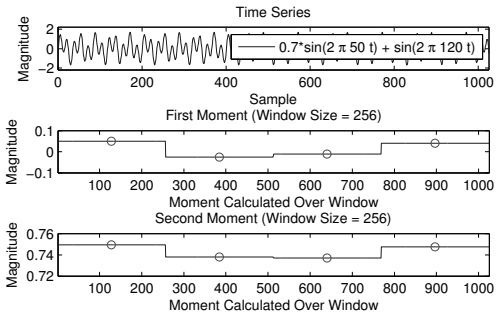
The data used to generate these plots is available in the accompanied CD under the *output* folder as described in Appendix E.

A.1 Verification of Analysis Functions

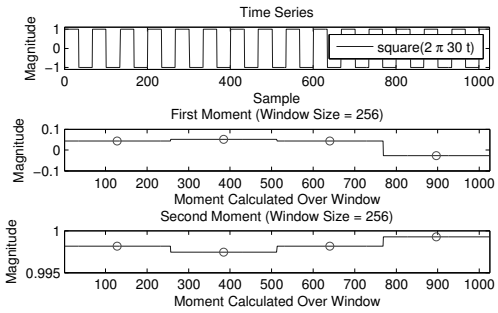
A.1.1 WSS Analysis Functions



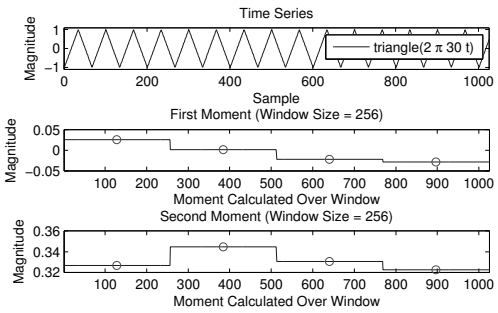
(a) Sinusoidal waveform verification.



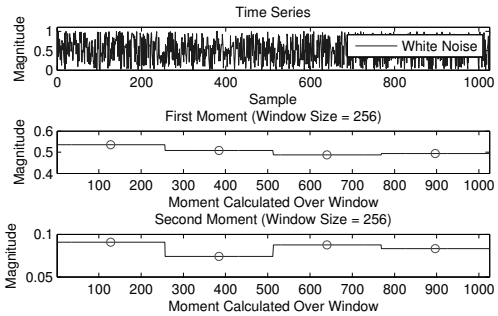
(b) Composite sinusoidal waveform verification.



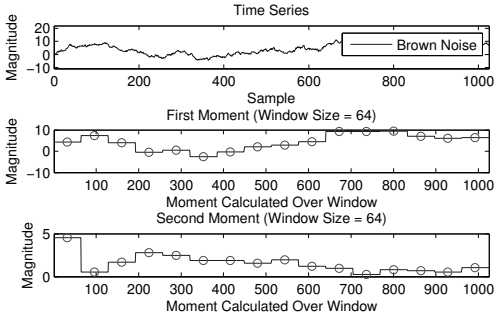
(c) Square waveform verification.



(d) Triangle waveform verification.



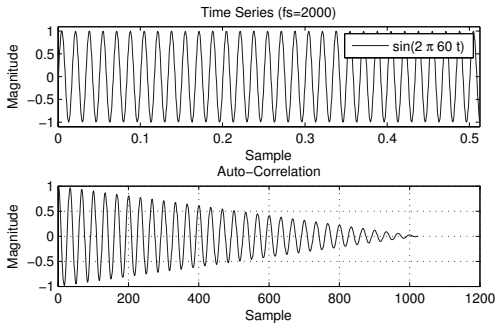
(e) White noise verification.



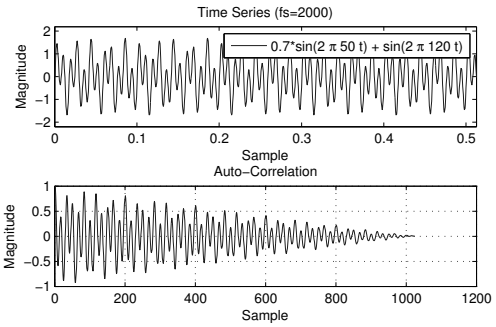
(f) Brown noise verification.

Fig. A.1: Verification of WSS analysis function using six test cases: (a) sinusoidal waveform, (b) composite sinusoidal waveform, (c) square wave, (d) triangle wave, (e) white noise, and (f) brown noise.

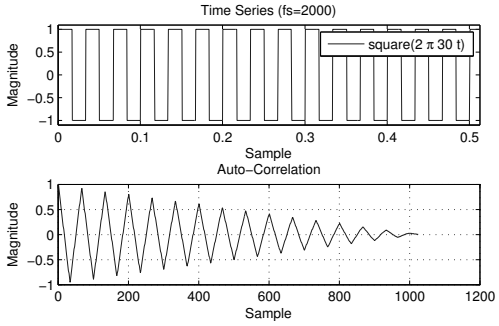
A.1.2 Auto-correlation Analysis Functions



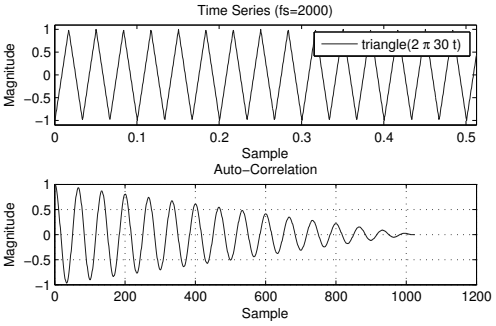
(a) Sinusoidal waveform verification.



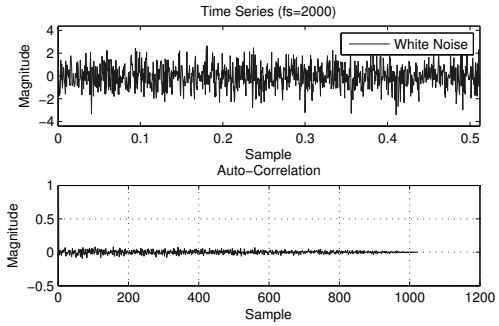
(b) Composite sinusoidal waveform verification.



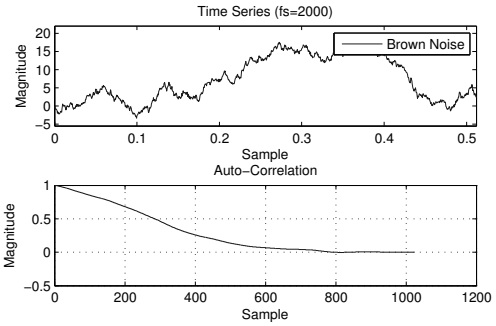
(c) Square waveform verification.



(d) Triangle waveform verification.



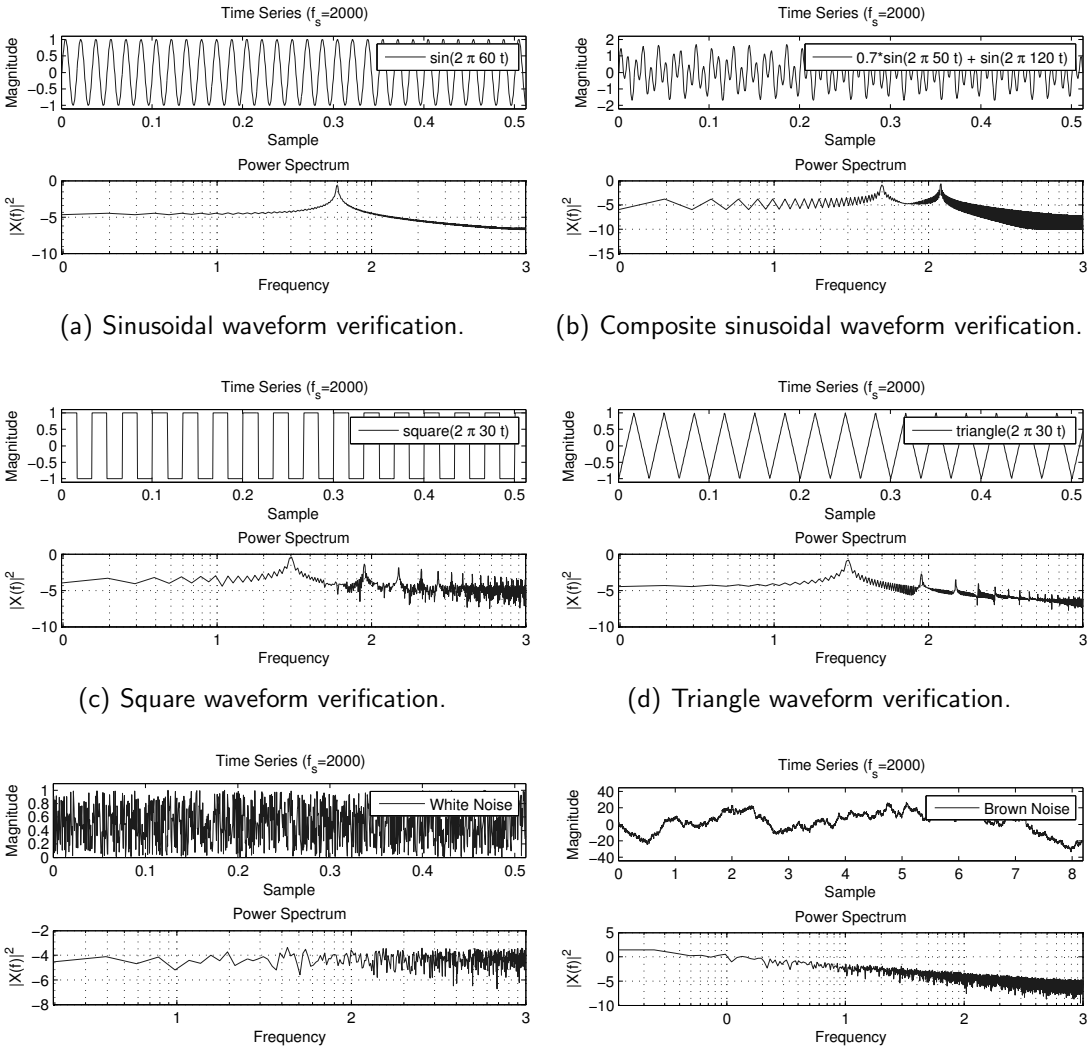
(e) White noise verification.



(f) Brown noise verification.

Fig. A.2: Verification of auto-correlation analysis function using six test cases: (a) sinusoidal waveform, (b) composite sinusoidal waveform, (c) square wave, (d) triangle wave, (e) white noise, and (f) brown noise.

A.1.3 Power Spectrum Analysis Functions



(a) Sinusoidal waveform verification.

(b) Composite sinusoidal waveform verification.

(c) Square waveform verification.

(d) Triangle waveform verification.

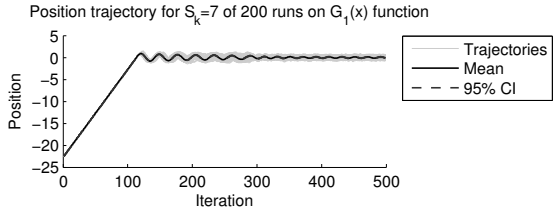
(e) White noise verification.

(f) Brown noise verification.

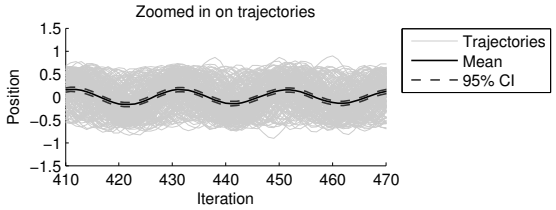
Fig. A.3: Verification of power spectrum analysis function using six test cases: (a) sinusoidal waveform, (b) composite sinusoidal waveform, (c) square wave, (d) triangle wave, (e) white noise, and (f) brown noise.

A.2 Sphere Function

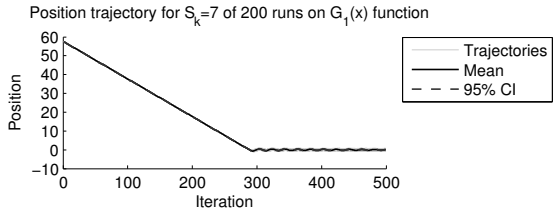
A.2.1 Ensemble Analysis



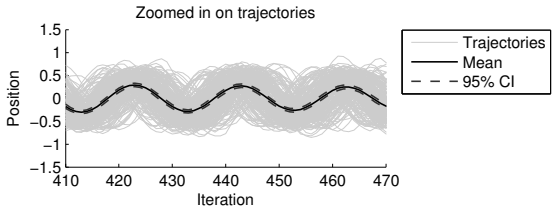
(a) Ensemble of $S_{x,7,1}$ for $G_1(\mathbf{x})$.



(b) Zoomed representation of Fig. A.4(a)



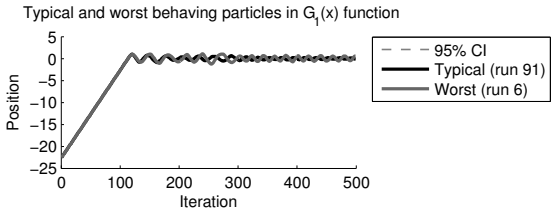
(c) Ensemble of $S_{x,7,2}$ for $G_1(\mathbf{x})$.



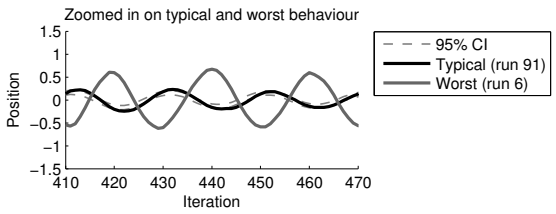
(d) Zoomed representation of Fig. A.4(c)

Fig. A.4: Ensemble of particle behaviours on Sphere function plotted against fitness for 200 runs. The mean and 95% confidence interval are shown. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.4(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.4(d).

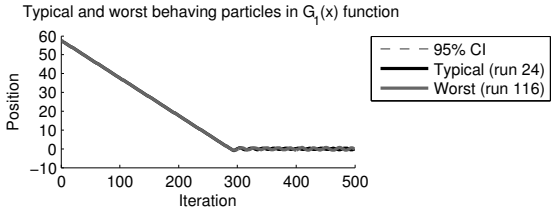
A.2.2 Typical Particle Trajectory



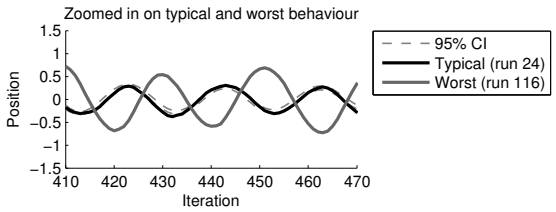
(a) Typical and worst particles $S_{x,7,1}$ for $G_1(\mathbf{x})$.



(b) Zoomed representation of Fig. A.5(a)



(c) Typical and worst particles $S_{x,7,2}$ for $G_1(\mathbf{x})$.



(d) Zoomed representation of Fig. A.5(c)

Fig. A.5: Identifying typical particle behaviours on Sphere function based on minimum MSE from the calculated trajectory mean, $\mu_{x,k,d}(n)$. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.5(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.5(c).

A.2.3 Time Series Extraction

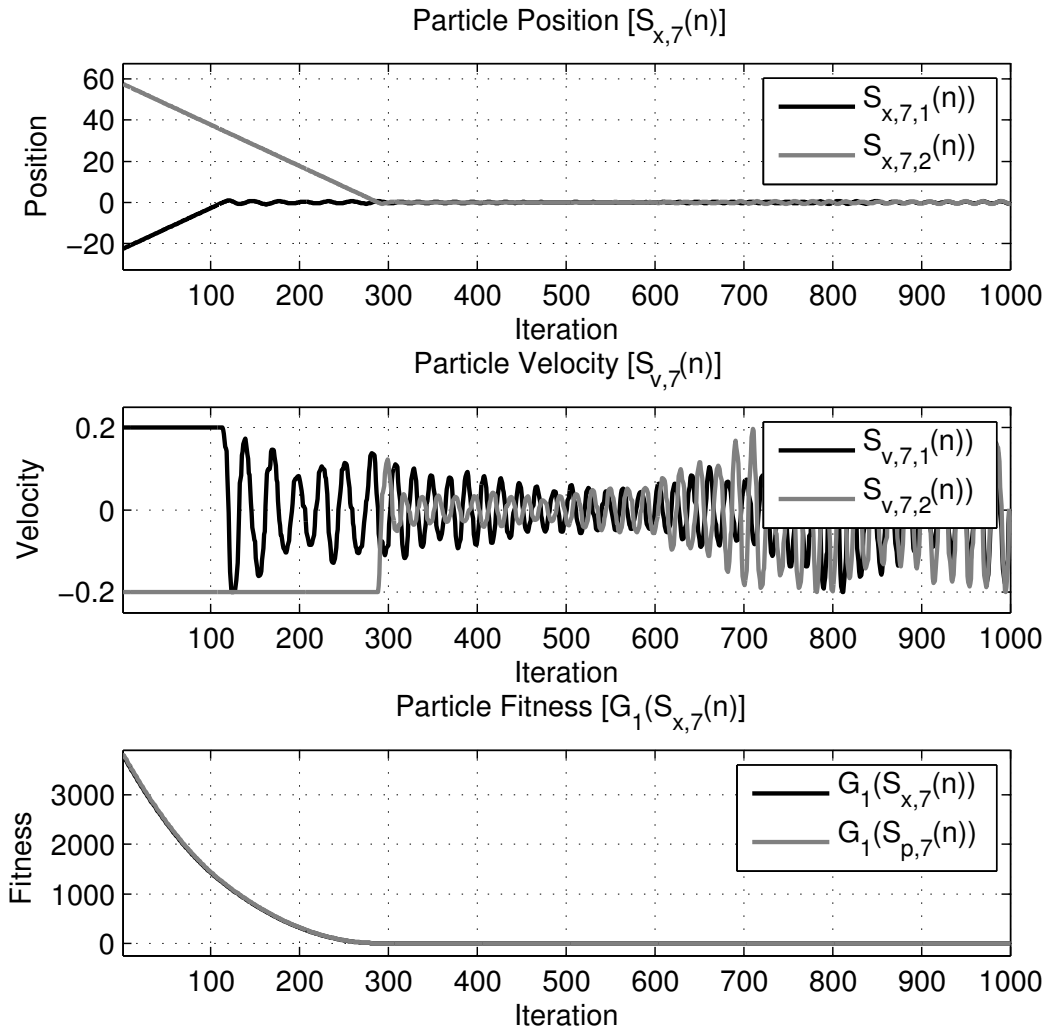
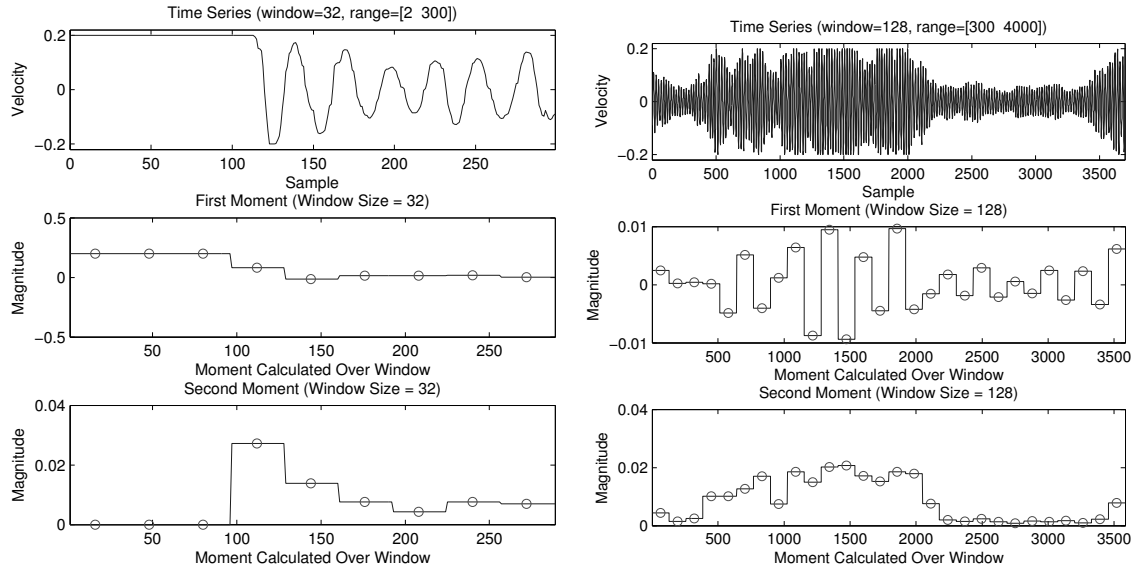


Fig. A.6: Extracted time series for particle on Sphere function.

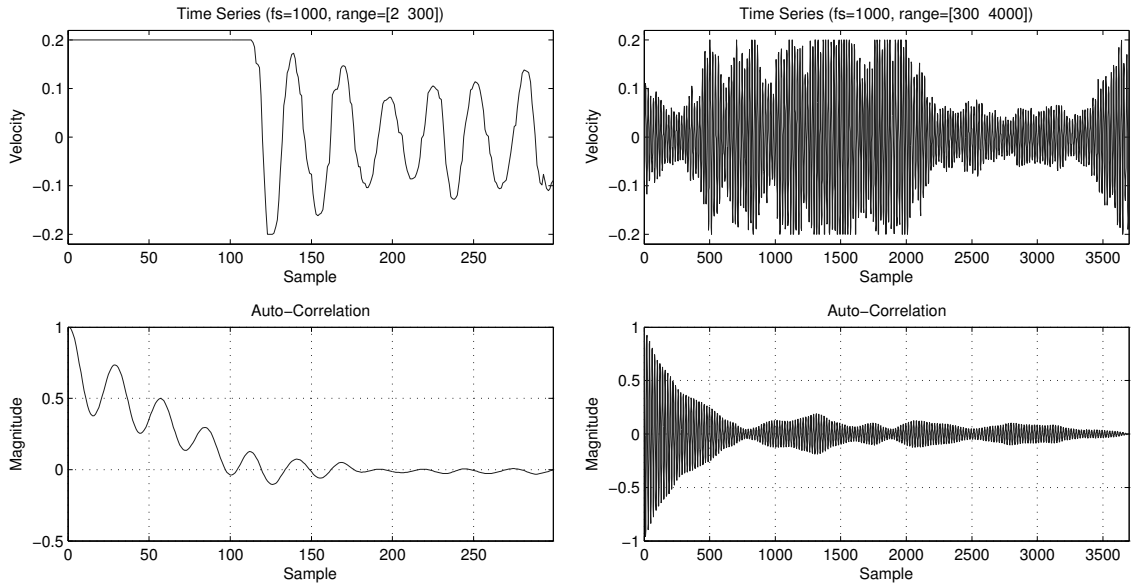
A.2.4 Time Domain Analysis - Moments



(a) Transient first and second moment for $G_1(\mathbf{x})$. (b) Steady state first and second moment for $G_1(\mathbf{x})$.

Fig. A.7: First and second moment of particle trajectory on the Sphere function. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory.

A.2.5 Time Domain Analysis - Auto-correlation

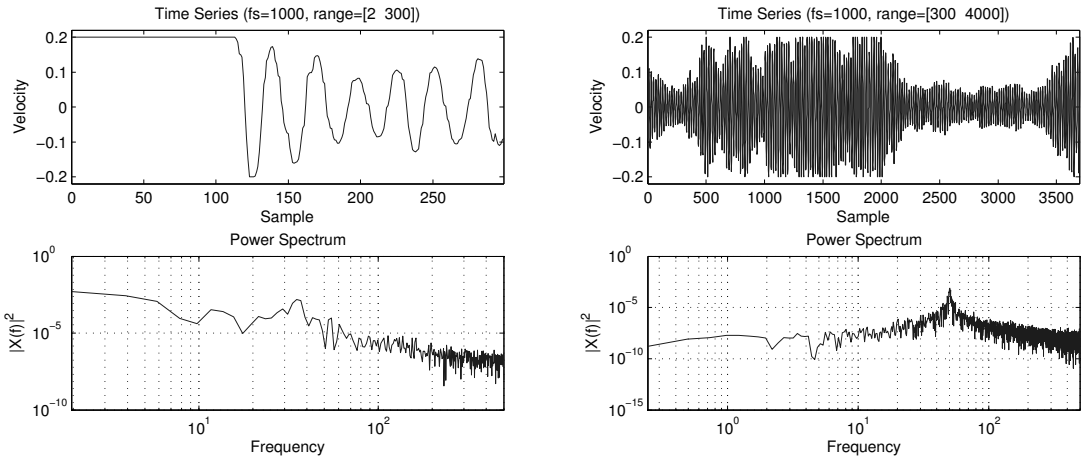


(a) Auto-correlation of trajectory for $G_1(\mathbf{x})$.

(b) Auto-correlation of trajectory for $G_1(\mathbf{x})$.

Fig. A.8: Auto-correlation of trajectory on the Sphere function. (a) Auto-correlation for transient portion of the trajectory. (b) Auto-correlation for the steady state portion of the trajectory.

A.2.6 Frequency Domain Analysis



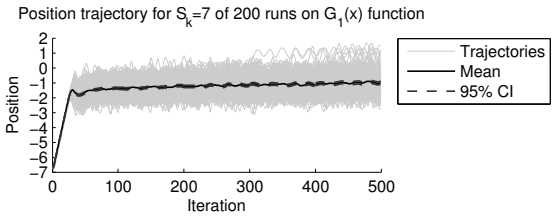
(a) Power spectrum of trajectory for $G_1(\mathbf{x})$.

(b) Power spectrum of trajectory for $G_1(\mathbf{x})$.

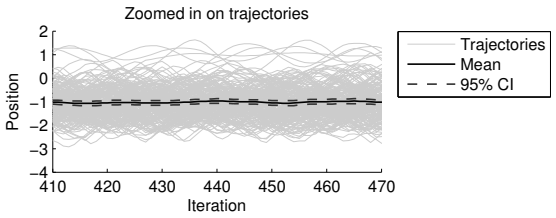
Fig. A.9: Power spectrum of trajectory on the Sphere function. (a) Power spectrum for transient portion of the trajectory. (b) Power spectrum for the steady state portion of the trajectory.

A.3 Rosenbrock Function

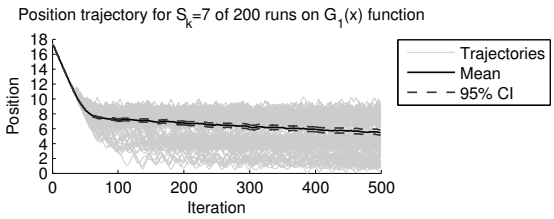
A.3.1 Ensemble Analysis



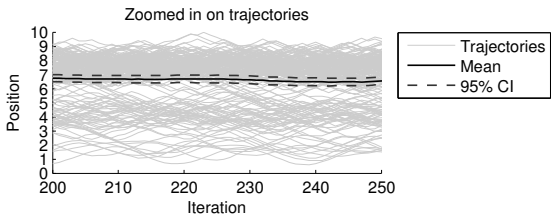
(a) Ensemble of $S_{x,7,1}$ for $G_2(\mathbf{x})$.



(b) Zoomed representation of Fig. A.10(a)



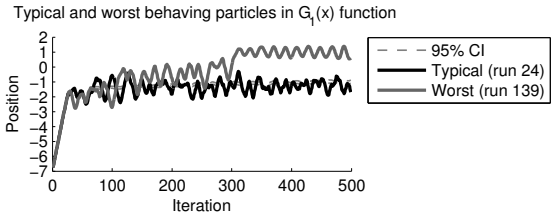
(c) Ensemble of $S_{x,7,2}$ for $G_2(\mathbf{x})$.



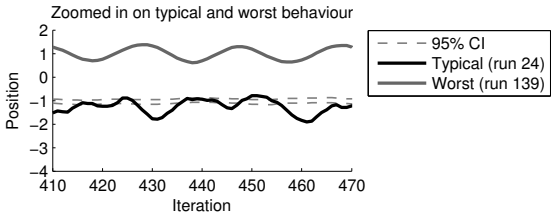
(d) Zoomed representation of Fig. A.10(c)

Fig. A.10: Ensemble of particle behaviours on Rosenbrock function plotted against fitness for 200 runs. The mean and 95% confidence interval are shown. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.10(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.10(d).

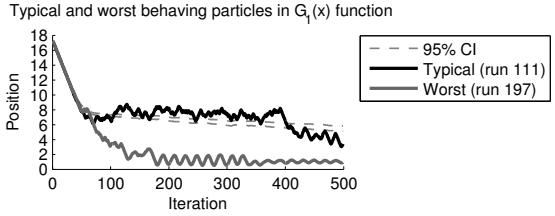
A.3.2 Typical Particle Trajectory



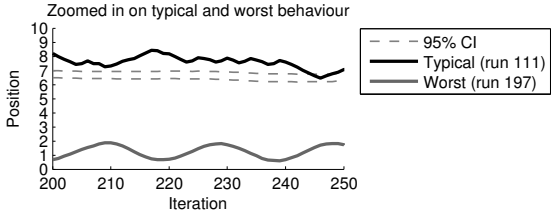
(a) Typical and worst particles $S_{x,7,1}$ for $G_2(\mathbf{x})$.



(b) Zoomed representation of Fig. A.11(a)



(c) Typical and worst particles $S_{x,7,2}$ for $G_2(\mathbf{x})$.



(d) Zoomed representation of Fig. A.11(c)

Fig. A.11: Identifying typical particle behaviours on Rosenbrock function based on minimum MSE from the calculated trajectory mean, $\mu_{x,k,d}(n)$. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.11(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.11(c).

A.3.3 Time Series Extraction

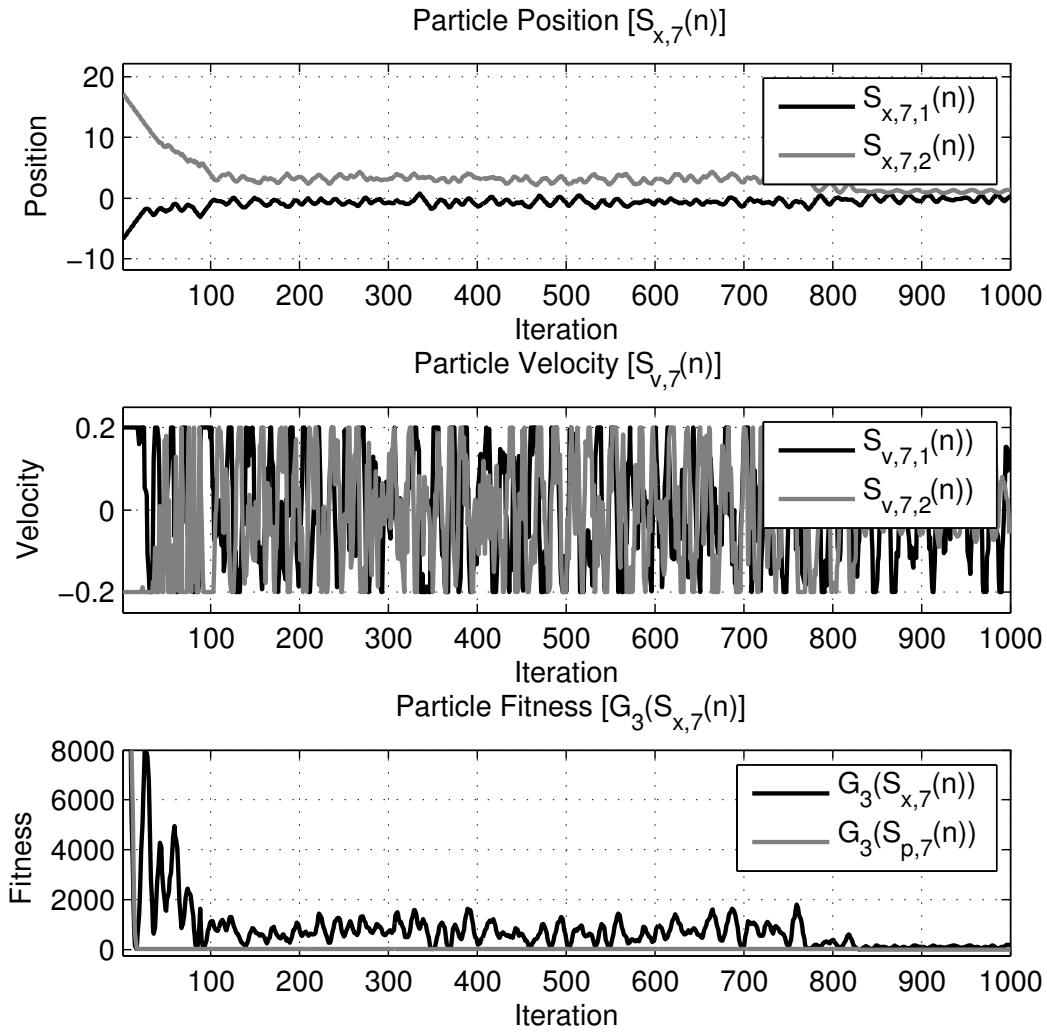
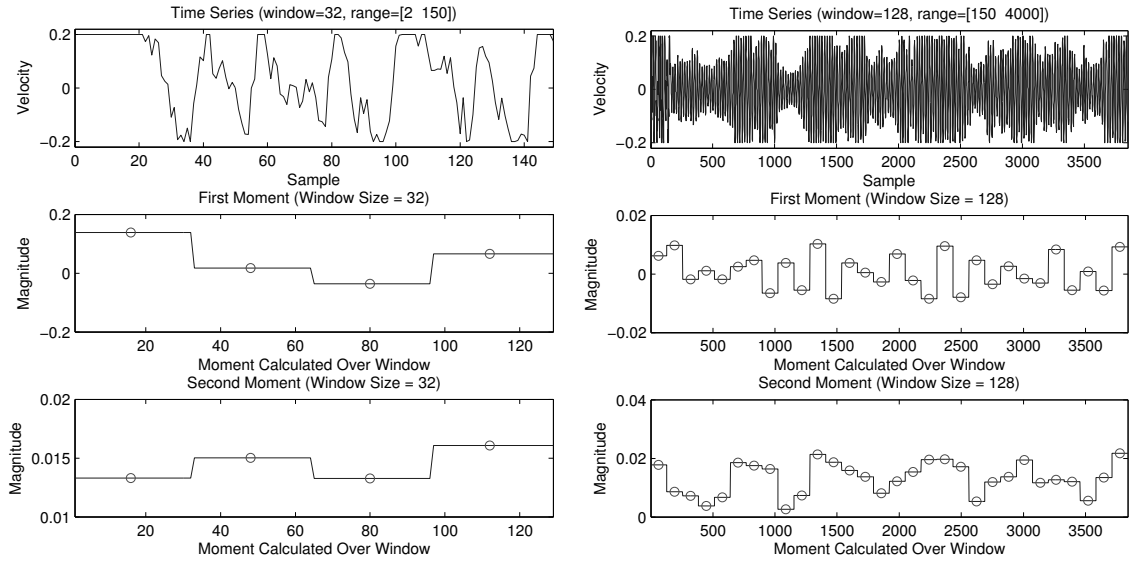


Fig. A.12: Extracted time series for particle on Rosenbrock function.

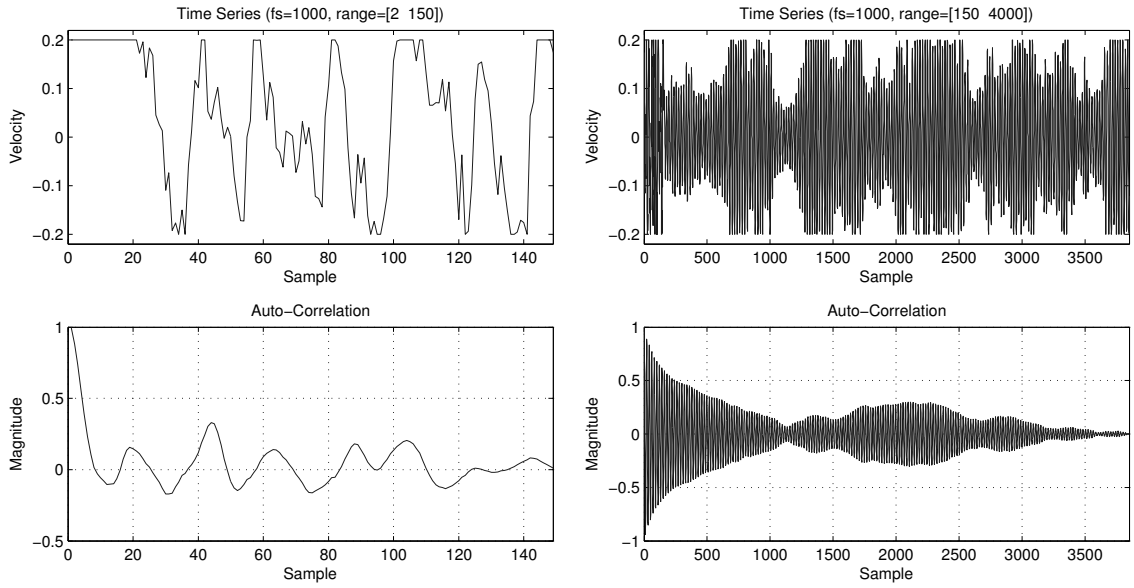
A.3.4 Time Domain Analysis - Moments



(a) Transient first and second moment for $G_2(\mathbf{x})$. (b) Steady state first and second moment for $G_2(\mathbf{x})$.

Fig. A.13: First and second moment of particle trajectory on the Rosenbrock function. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory.

A.3.5 Time Domain Analysis - Auto-correlation

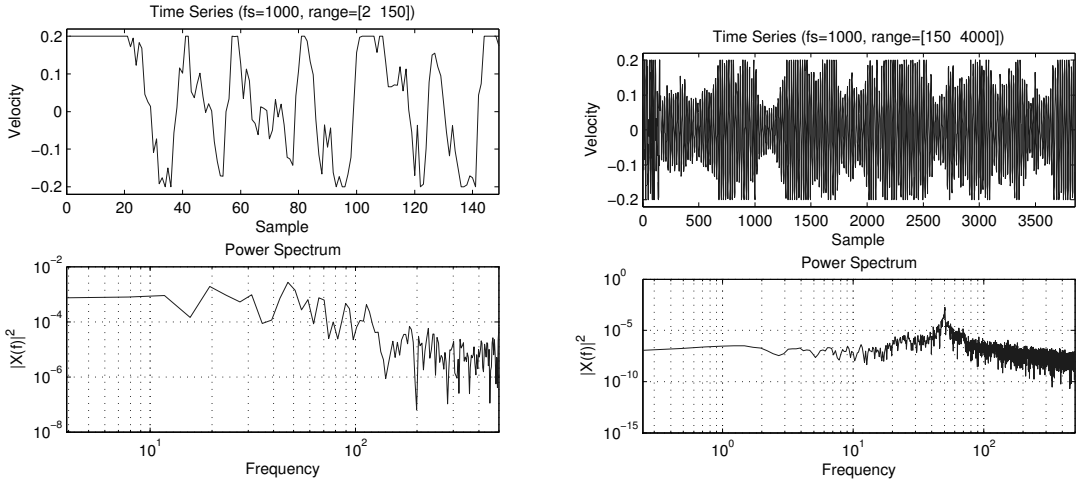


(a) Auto-correlation of trajectory for $G_2(\mathbf{x})$.

(b) Auto-correlation of trajectory for $G_2(\mathbf{x})$.

Fig. A.14: Auto-correlation of trajectory on the Rosenbrock function. (a) Auto-correlation for transient portion of the trajectory. (b) Auto-correlation for the steady state portion of the trajectory.

A.3.6 Frequency Domain Analysis



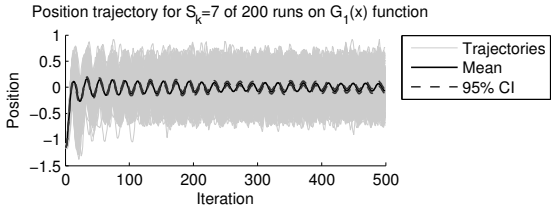
(a) Power spectrum of trajectory for $G_2(\mathbf{x})$.

(b) Power spectrum of trajectory for $G_2(\mathbf{x})$.

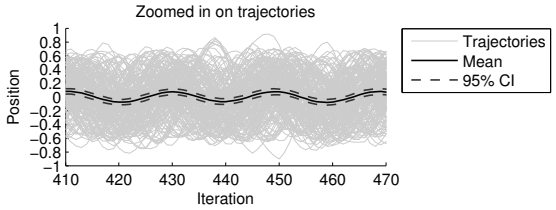
Fig. A.15: Power spectrum of trajectory on the Rosenbrock function. (a) Power spectrum for transient portion of the trajectory. (b) Power spectrum for the steady state portion of the trajectory.

A.4 Rastrigin Function

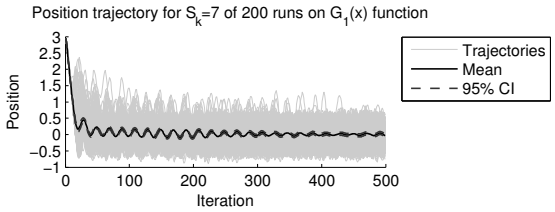
A.4.1 Ensemble Analysis



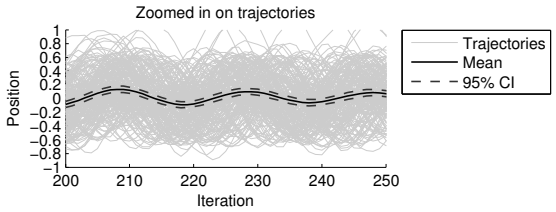
(a) Ensemble of $S_{x,7,1}$ for $G_3(\mathbf{x})$.



(b) Zoomed representation of Fig. A.16(a)



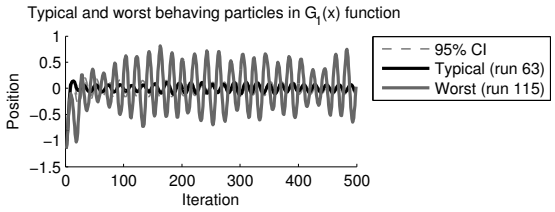
(c) Ensemble of $S_{x,7,2}$ for $G_3(\mathbf{x})$.



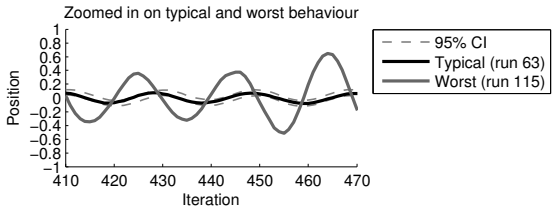
(d) Zoomed representation of Fig. A.16(c)

Fig. A.16: Ensemble of particle behaviours on Rastrigin function plotted against fitness for 200 runs. The mean and 95% confidence interval are shown. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.16(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.16(d).

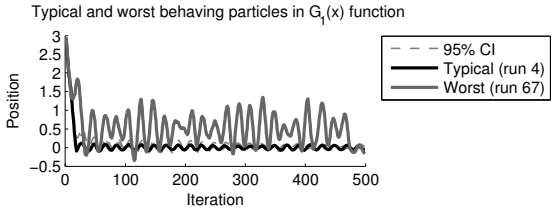
A.4.2 Typical Particle Trajectory



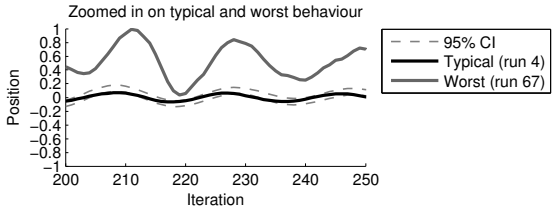
(a) Typical and worst particles $S_{x,7,1}$ for $G_3(\mathbf{x})$.



(b) Zoomed representation of Fig. A.17(a)



(c) Typical and worst particles $S_{x,7,2}$ for $G_3(\mathbf{x})$.



(d) Zoomed representation of Fig. A.17(c)

Fig. A.17: Identifying typical particle behaviours on Rastrigin function based on minimum MSE from the calculated trajectory mean, $\mu_{x,k,d}(n)$. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.17(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.17(c).

A.4.3 Time Series Extraction

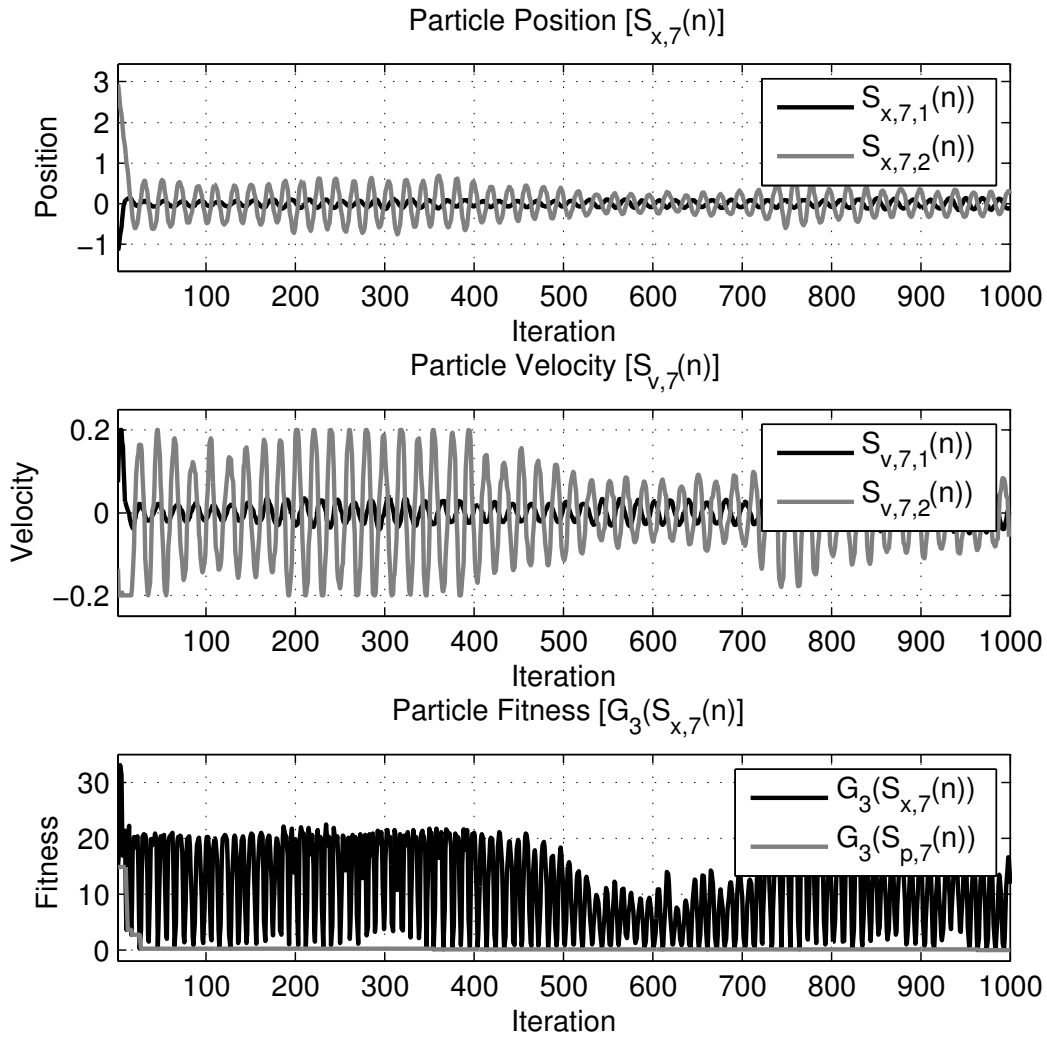
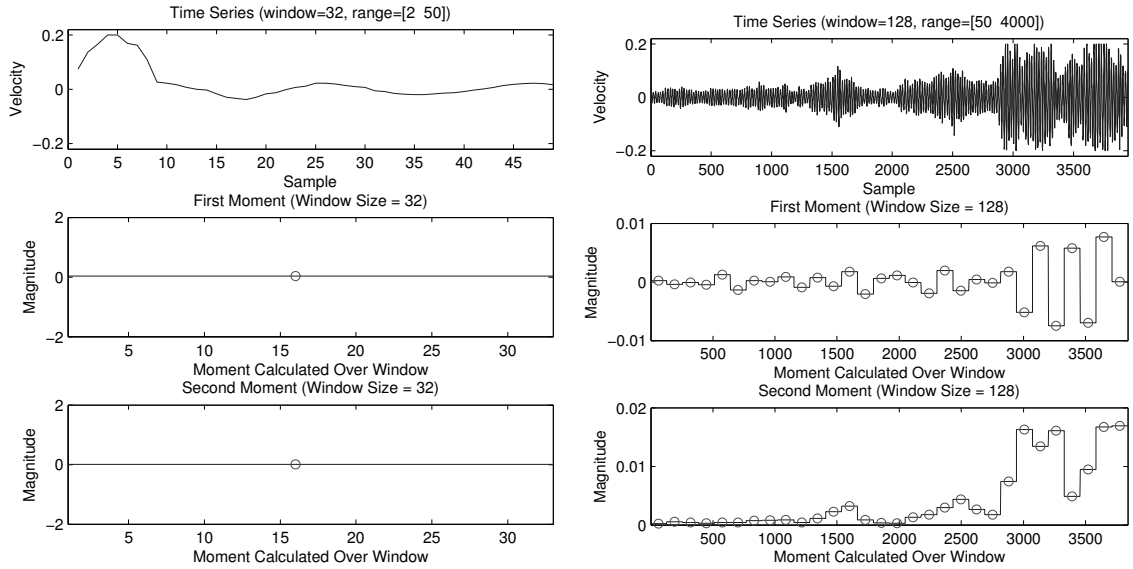


Fig. A.18: Time series showing the particles position, velocity, and fitness over time in the Rastrigin function. (a) The particles position over time along the 2 dimensions being optimized. (b) The velocity of the particle over time along the 2 dimensions being optimized. (c) The fitness of the particle as it converges to the global optimum when the fitness equals zero. This plot highlights the monotonically decreasing fitness of the best position and contrasts the variation in the current positions fitness as the particle explores many of the local solution spaces.

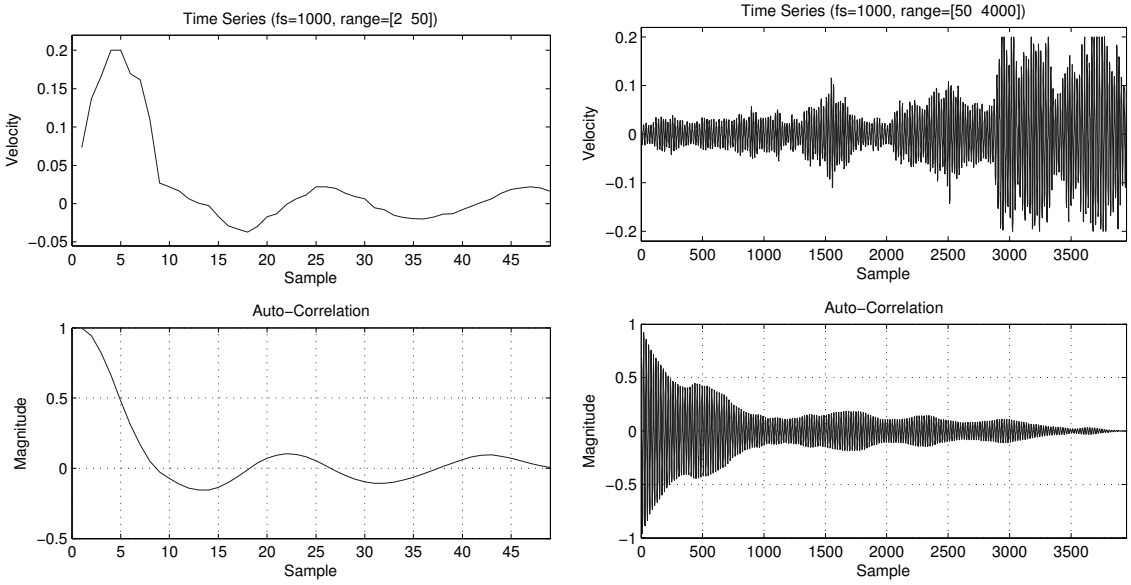
A.4.4 Time Domain Analysis - Moments



(a) Transient first and second moment for $G_3(\mathbf{x})$. (b) Steady state first and second moment for $G_3(\mathbf{x})$.

Fig. A.19: First and second moment of particle trajectory on the Rastrigin function. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory.

A.4.5 Time Domain Analysis - Auto-correlation

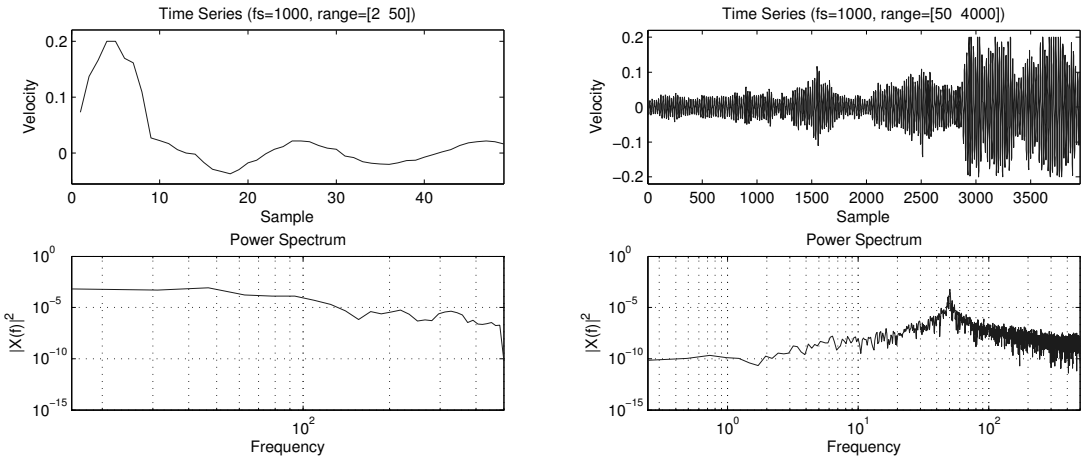


(a) Auto-correlation of trajectory for $G_3(\mathbf{x})$.

(b) Auto-correlation of trajectory for $G_3(\mathbf{x})$.

Fig. A.20: Auto-correlation of trajectory on the Rastrigin function. (a) Auto-correlation for transient portion of the trajectory. (b) Auto-correlation for the steady state portion of the trajectory.

A.4.6 Frequency Domain Analysis



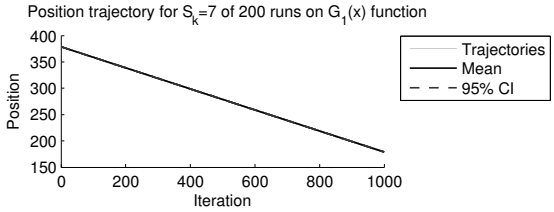
(a) Power spectrum of trajectory for $G_3(\mathbf{x})$.

(b) Power spectrum of trajectory for $G_3(\mathbf{x})$.

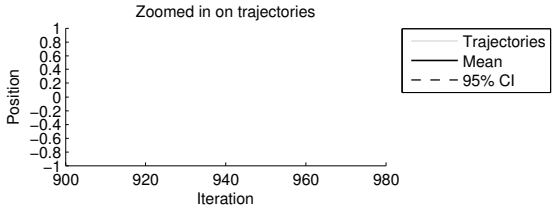
Fig. A.21: Power spectrum of trajectory on the Rastrigin function. (a) Power spectrum for transient portion of the trajectory. (b) Power spectrum for the steady state portion of the trajectory.

A.5 Griewank Function

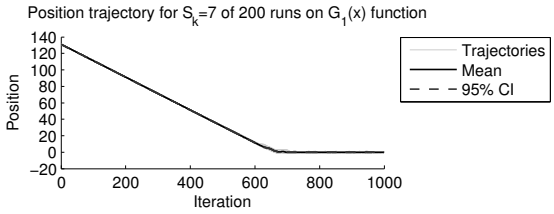
A.5.1 Ensemble Analysis



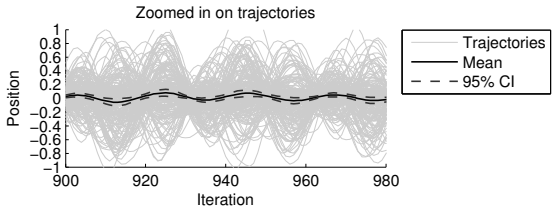
(a) Ensemble of $S_{x,7,1}$ for $G_4(\mathbf{x})$.



(b) Zoomed representation of Fig. A.22(a)



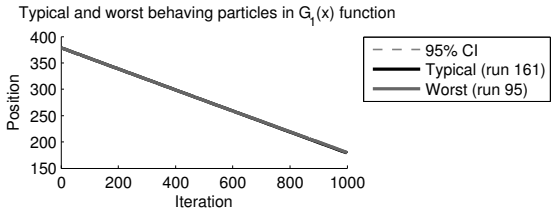
(c) Ensemble of $S_{x,7,2}$ for $G_4(\mathbf{x})$.



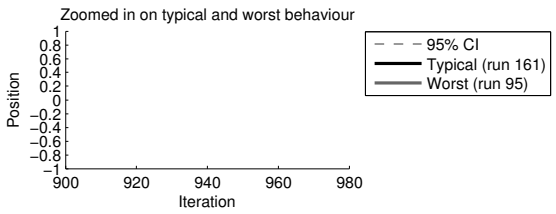
(d) Zoomed representation of Fig. A.22(c)

Fig. A.22: Ensemble of particle behaviours on Griewank function plotted against fitness for 200 runs. The mean and 95% confidence interval are shown. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.22(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.22(d).

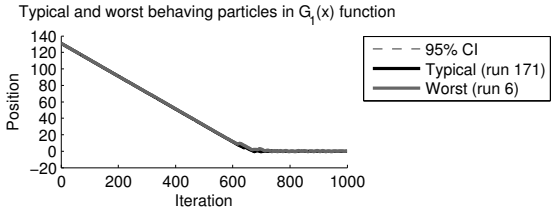
A.5.2 Typical Particle Trajectory



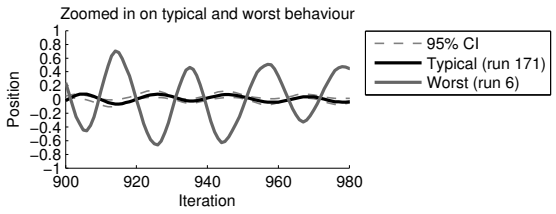
(a) Typical and worst particles $S_{x,7,1}$ for $G_4(\mathbf{x})$.



(b) Zoomed representation of Fig. A.23(a)



(c) Typical and worst particles $S_{x,7,2}$ for $G_4(\mathbf{x})$.



(d) Zoomed representation of Fig. A.23(c)

Fig. A.23: Identifying typical particle behaviours on Griewank function based on minimum MSE from the calculated trajectory mean, $\mu_{x,k,d}(n)$. (a) Position, $S_{x,7,1}$, along $d = 1$ dimension. (b) Zoomed in representation of Fig. A.23(a). (c) Position, $S_{x,7,2}$, along $d = 2$ dimension. (d) Zoomed in representation of Fig. A.23(c).

A.5.3 Time Series Extraction

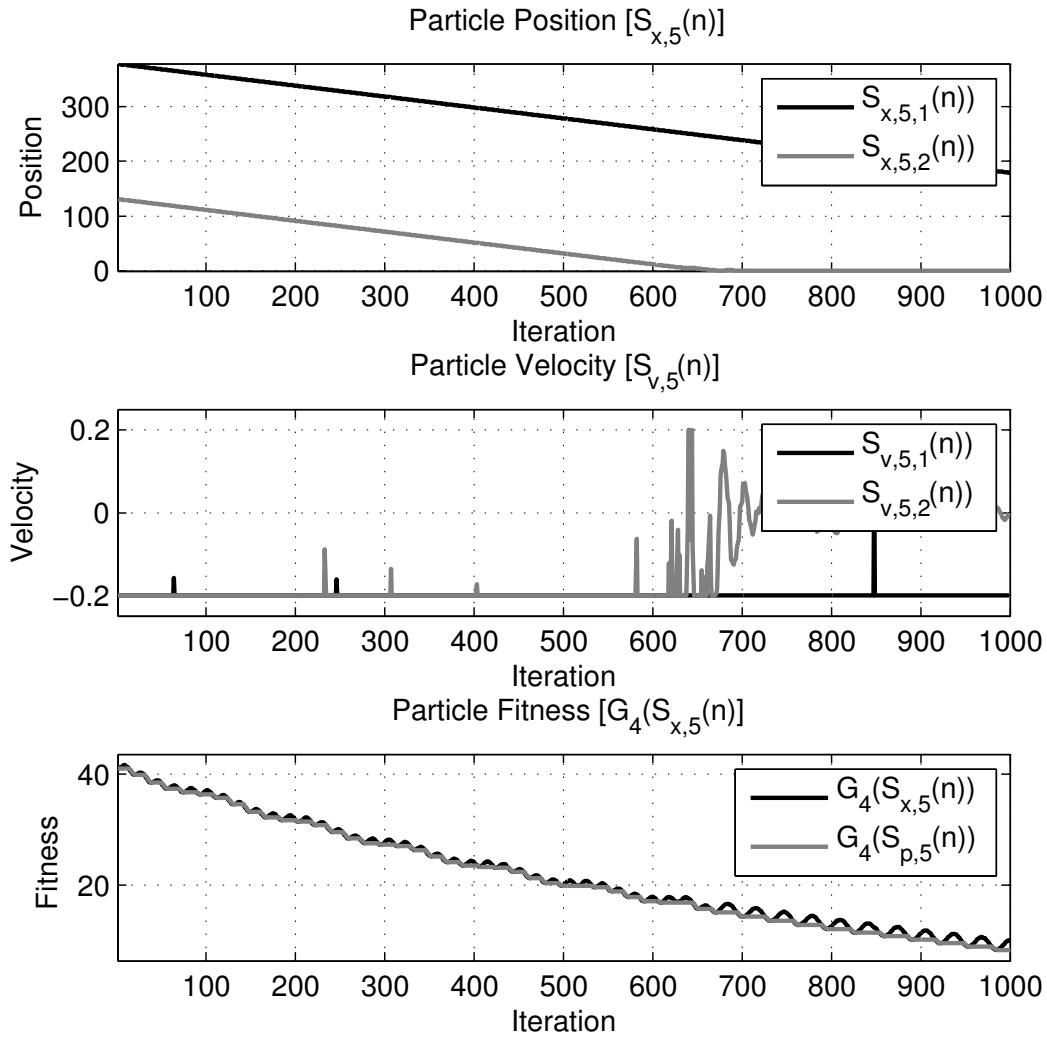
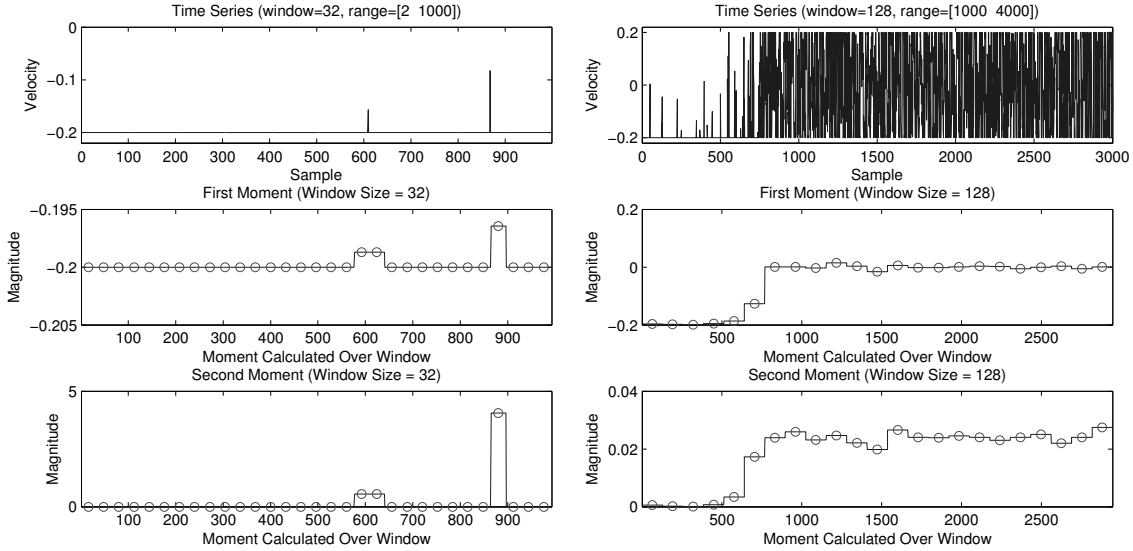


Fig. A.24: Extracted time series for particle on Griewank function.

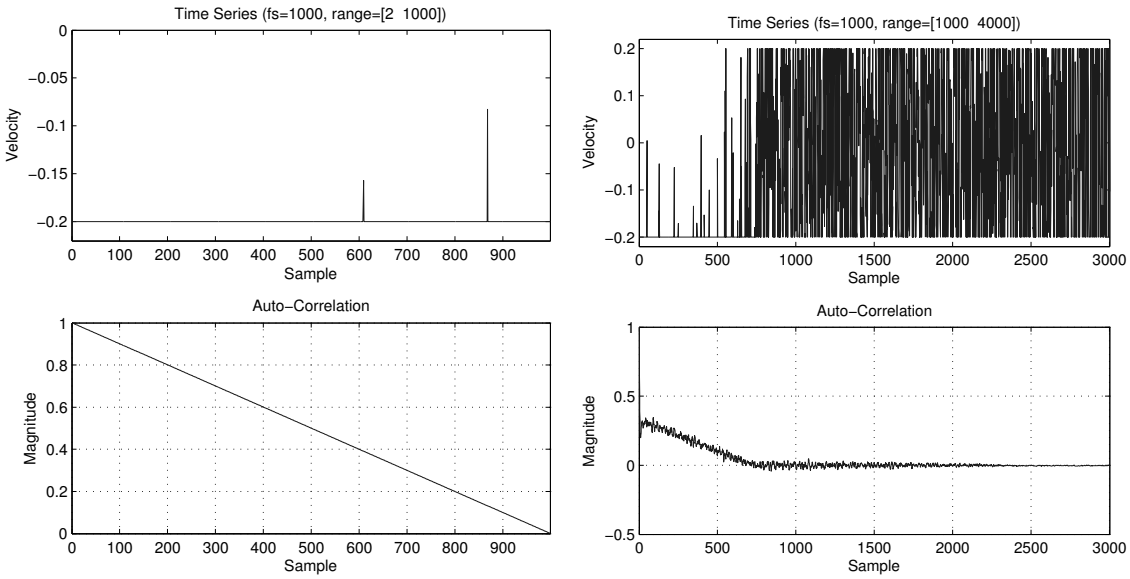
A.5.4 Time Domain Analysis - Moments



(a) Transient first and second moment for $G_4(\mathbf{x})$. (b) Steady state first and second moment for $G_4(\mathbf{x})$.

Fig. A.25: First and second moment of particle trajectory on the Griewank function. (a) Moments calculated for transient portion of the trajectory. (b) Moments calculated for the steady state portion of the trajectory.

A.5.5 Time Domain Analysis - Auto-correlation

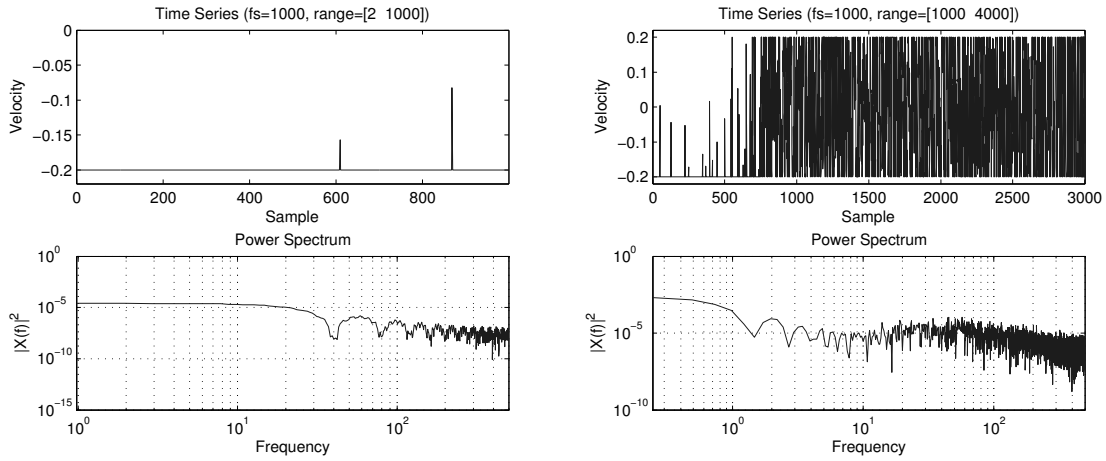


(a) Auto-correlation of trajectory for $G_4(\mathbf{x})$.

(b) Auto-correlation of trajectory for $G_4(\mathbf{x})$.

Fig. A.26: Auto-correlation of trajectory on the Griewank function. (a) Auto-correlation for transient portion of the trajectory. (b) Auto-correlation for the steady state portion of the trajectory.

A.5.6 Frequency Domain Analysis



(a) Power spectrum of trajectory for $G_4(\mathbf{x})$.

(b) Power spectrum of trajectory for $G_4(\mathbf{x})$.

Fig. A.27: Power spectrum of trajectory on the Griewank function. (a) Power spectrum for transient portion of the trajectory. (b) Power spectrum for the steady state portion of the trajectory.

Appendix B

Software

This chapter outlines the basic instructions to run the optimization algorithm and the scheduler described in Sec. 4.1 and 4.2 respectively. The source code described in this appendix is included in the DVD accompanied this thesis labeled Disc 1. Matlab 2010b or later is recommended for these applications. Note that in 2012b or later versions there will be warnings about the random number generation as those functions are going to be deprecated as of version 2013b.

B.1 Running PSO Algorithm on Test Functions

1. Create a configuration file like the one shown in Appendix C.1. For convinience, a sample file is provided in *sims* folder with the source code. For this purpose, ignore all the scheduler-specific parameters in the configuration file as they will not be processed by the software.
2. Select the appropriate parameters to use for PSO. Pay special attention to the following:
   (a) The `rootFolder` should point to the directory where the config file is saved.
   (b) Set `topo evaluate` to the desired topology from those available in the folder *sims/pso/topologies*.
   (c) Set `phi evaluate` to the desired personal and social weights file from the folder *sims/pso/weights*.

(d) Set `cf evaluate` to the desired function to evaluate. The four functions selected in the thesis as well as the scheduler function are included in *sims/pso/function*

3. Open the file `main_test.m` in Matlab.
4. Select the appropriate configuration file on Line 9 of the file.
5. Run the script.
6. The results will be displayed on the console and saved to files in the root folder previously defined.

B.2 Running Scheduling Algorithm

1. If a custom set of tasks is to be evaluated, create a file with four columns for *task number*, *start time*, *computation time*, and *relative deadline* and define as many tasks as needed. A sample is provided in the folder *sims*. Alternatively, if this file is not defined, the software will generate random sets of tasks.
2. Create a configuration file like the one shown in Appendix C.1. For convinience, a sample file is provided in *sims* folder with the source code.
3. Select the appropriate parameters to use for PSO. Pay special attention to the following:
   (a) The `rootFolder` should point to the directory where the config file is saved.
   (b) Set `topo evaluate` to the desired topology from those available in the folder *sims/pso/topologies*.
   (c) Set `phi evaluate` to the desired personal and social weights file from the folder *sims/pso/weights*.
   (d) Set `cf evaluate` to the `@minTotalTardinessN`
   (e) Set the number of tasks to match the number of dimensions to satisfy the RK encoding used in the scheduler.
4. Open the file `main_sched.m` in Matlab.
5. Select the appropriate configuration file on Line 9 of the file.

6. Run the script.
7. The results will be displayed on the console and saved to files in the root folder previously defined.

Appendix C

Sample Output

C.1 Sample Configuration File

```
###
# Scheduler Configuration
###
rootFolder          /control-p1-t10-130512/
nProcessors         1
nTasks              20
seed                20
saveTraj            1
saveMap             1
printProgress       1

###
# Test Number
###
nRun                10
iRun                1

###
# Particple Swarm Optimization
###
pso filename        folder
pso nParticles      30
pso nDim            20
pso nIter           1000

###
# Mutation
###
```

```
mutation enabled    1
mutation min        0.1
mutation max        0.4


###
# PSO Neighborhood Topology
###
topo name           gBest
topo evaluate       @gBest
topo size           5

###
# PSO Weights
###
phi name            phiShEb99
phi evaluate        @phiShEb99

###
# Cost Function
###
cf name             minTotalTardinessN
cf evaluate         @minTotalTardinessN
cf xMax             1
cf xMin             0
cf vMax             1
cf wMin             0.729
cf wMax             0.729
cf wEph             1

###
# Task Properties
###
taskProp filename   tasks.txt
taskProp SRange     [0,25]
taskProp CRange     [1,5]
taskProp dRange     [10,15]
taskProp randDist   @uniform
taskProp Rmin       1
taskProp Rmax       1
```

C.2 Sample Particle Trajectory Recorded

```
-4.014363 -0.104207 -6.267440 +0.001778 -0.001223 -0.003677 +1.646735 +0000.009948 5
-3.905206 +0.109157 -6.267440 +0.000344 -0.001434 -0.003677 +1.726154 +0000.009948 5
-3.705206 +0.200000 -6.267440 -0.001248 -0.001592 -0.003677 +1.848762 +0000.009948 5
-3.505206 +0.200000 -6.267440 -0.002934 -0.001687 -0.003677 +1.937687 +0000.009948 5
```

```
-3.305206 +0.200000 -6.267440 -0.004485 -0.001551 -0.003677 +1.989371 +0000.009948 5
-3.145616 +0.159590 -6.267440 -0.005922 -0.001437 -0.003677 +2.002457 +0000.009948 5
-2.945616 +0.200000 -6.267440 -0.006637 -0.000716 -0.003677 +1.983016 +0000.009948 5
-2.841239 +0.104377 -6.267440 -0.007089 -0.000452 -0.003677 +1.957238 +0000.009948 5
-2.641239 +0.200000 -6.267440 -0.006848 +0.000241 -0.003677 +1.879147 +0000.009948 5
-2.511685 +0.129554 -6.267440 -0.006049 +0.000799 -0.003677 +1.809652 +0000.009948 5
-2.659421 -0.147736 -6.267440 -0.005036 +0.001013 -0.003677 +1.887753 +0000.009948 5
-2.859421 -0.200000 -6.267440 -0.003526 +0.001511 -0.003677 +1.962494 +0000.009948 5
-3.059421 -0.200000 -6.267440 -0.001755 +0.001770 -0.003677 +1.998965 +0000.009948 5
-3.259421 -0.200000 -6.267440 +0.000005 +0.001761 -0.003677 +1.995722 +0000.009948 5
-3.459421 -0.200000 -6.267440 +0.001678 +0.001673 -0.003677 +1.952907 +0000.009948 5
```

C.3 Sample Particle Schedule Recorded

```
1 2 3 5 4 6 8 7 11 10 14 13 18 15 9 12 17 16 19 20 -1.711947 +0.018196 -1.730142 -1.403264 -0.004159
-1.399105 -1.132604 +0.014244 -1.146848 -0.935814 -0.090527 -0.845286 -1.010882 -0.252769 -0.758113
-0.669972 +0.067833 -0.737805 -0.511194 +0.034281 -0.545475 -0.560023 -0.024575 -0.535448 +1.000000
+0.936448 -0.196329 +0.087901 -0.034603 +0.122504 +0.057896 -0.125689 +0.183586 +1.014673 +0.379973
+0.634700 +0.573443 +0.035106 +0.538337 +0.507801 -0.099345 +0.607146 +0.838776 -0.010777 +0.849553
+1.127876 +0.070212 +1.057664 +1.024255 +0.087409 +0.936847 +0.591890 -0.305303 +0.897193 +1.160588
+0.017327 +1.143261 +1.857586 +0.029484 +1.828102 +340.000000 +258.000000 1 1 2 3 5 4 6 7 8 9 10 11
13 18 14 12 15 17 16 19 20 -1.722269 -0.010322 -1.730142 -1.398802 +0.004461 -1.399105 -1.128910
+0.003693 -1.146848 -0.908924 +0.026890 -0.845286 -1.054502 -0.043620 -0.758113 -0.700786 -0.030814
-0.737805 -0.559193 -0.047999 -0.545475 -0.514931 +0.045093 -0.535448 +0.000000 -1.000000 -0.196329
+0.138441 +0.050540 +0.122504 +0.292757 +0.234861 +0.183586 +0.793380 -0.221294 +0.634700 +0.486236
-0.087207 +0.538337 +0.547710 +0.039910 +0.607146 +0.837157 -0.001619 +0.849553 +1.056757 -0.071119
+1.057664 +1.055576 +0.031321 +0.936847 +0.537353 -0.054537 +0.897193 +1.147278 -0.013310 +1.143261
+2.068019 +0.210434 +1.828102 +328.000000 +258.000000 1 1 2 3 9 4 5 6 7 8 12 10 13 11 14 15 18 16 17
19 20 -1.686303 +0.035966 -1.730142 -1.395483 +0.003319 -1.399105 -1.137330 -0.008420 -1.146848
-0.878531 +0.030392 -0.845286 -0.809273 +0.245229 -0.758113 -0.751047 -0.050261 -0.737805 -0.587303
-0.028111 -0.545475 -0.503220 +0.011710 -0.535448 -1.000000 -1.000000 -0.196329 +0.138560 +0.000119
+0.122504 +0.421028 +0.128270 +0.183586 -0.029351 -0.822731 +0.634700 +0.411782 -0.074454 +0.538337
+0.654555 +0.106844 +0.607146 +0.876586 +0.039429 +0.849553 +0.940080 -0.116676 +1.057664 +1.058346
+0.002770 +0.936847 +0.924653 +0.387300 +0.897193 +1.129001 -0.018277 +1.143261 +2.105953 +0.037934
+1.828102 +400.000000 +258.000000 1 1 2 3 9 4 6 7 8 12 11 10 13 14 15 16 5 17 19 18 20 -1.677092
+0.009210 -1.730142 -1.399695 -0.004212 -1.399105 -1.140290 -0.002960 -1.146848 -1.048658 -0.170127
-0.845286 +1.000000 -0.967752 -0.758113 -0.742168 +0.008879 -0.737805 -0.578712 +0.008591 -0.545475
-0.516460 -0.013240 -0.535448 -1.056820 -0.056820 -0.196329 +0.105871 -0.032689 +0.122504 +0.069315
-0.351713 +0.183586 -0.033633 -0.004282 +0.634700 +0.433191 +0.021409 +0.538337 +0.622393 -0.032161
+0.607146 +0.888731 +0.012145 +0.849553 +0.959525 +0.019444 +1.057664 +1.090431 +0.032085 +0.936847
+1.210973 +0.286320 +0.897193 +1.137500 +0.008499 +1.143261 +1.977460 -0.128493 +1.828102
+400.000000 +258.000000 1 1 2 3 4 6 7 8 9 5 11 10 13 14 15 18 17 12 16 19 20 -1.682313 -0.005220
-1.730142 -1.400169 -0.000474 -1.399105 -1.149109 -0.008819 -1.146848 -0.976705 +0.071953 -0.845286
+0.016025 -0.983975 -0.758113 -0.676736 +0.065432 -0.737805 -0.586741 -0.008029 -0.545475 -0.508109
+0.008351 -0.535448 -0.056820 +1.000000 -0.196329 +0.091038 -0.014833 +0.122504 +0.089395 +0.020080
+0.183586 +0.966367 +1.000000 +0.634700 +0.430313 -0.002878 +0.538337 +0.568149 -0.054245 +0.607146
+0.862726 -0.026005 +0.849553 +1.056633 +0.097109 +1.057664 +0.952601 -0.137830 +0.936847 +0.943928
```

```
-0.267045 +0.897193 +1.145785 +0.008285 +1.143261 +1.694181 -0.283279 +1.828102 +286.000000
+258.000000 1 1 2 3 5 4 6 7 8 10 11 9 13 14 18 15 17 12 16 19 20 -1.734668 -0.052355 -1.730142
-1.396902 +0.003267 -1.399105 -1.143459 +0.005650 -1.146848 -0.927169 +0.049536 -0.845286 -0.983975
-1.000000 -0.758113 -0.657318 +0.019418 -0.737805 -0.567754 +0.018987 -0.545475 -0.520283 -0.012175
-0.535448 +0.210498 +0.267318 -0.196329 +0.115279 +0.024241 +0.122504 +0.156900 +0.067505 +0.183586
+1.087284 +0.120917 +0.634700 +0.424847 -0.005466 +0.538337 +0.575678 +0.007530 +0.607146 +0.833404
-0.029322 +0.849553 +1.096835 +0.040201 +1.057664 +0.971008 +0.018407 +0.936847 +0.724085 -0.219843
+0.897193 +1.145702 -0.000083 +1.143261 +2.059815 +0.365633 +1.828102 +306.000000 +258.000000 1 1 5
2 3 4 6 7 8 9 12 10 11 13 14 15 18 16 17 19 20 -1.725297 +0.009371 -1.730142 -1.397830 -0.000928
-1.399105 -1.128430 +0.015029 -1.146848 -0.843748 +0.083422 -0.845286 -1.521923 -0.537949 -0.758113
-0.649022 +0.008296 -0.737805 -0.561269 +0.006486 -0.545475 -0.527155 -0.006872 -0.535448 -0.379427
-0.589925 -0.196329 +0.140005 +0.024726 +0.122504 +0.364153 +0.207253 +0.183586 +0.087284 -1.000000
+0.634700 +0.476079 +0.051232 +0.538337 +0.636290 +0.060611 +0.607146 +0.862315 +0.028911 +0.849553
+1.033211 -0.063624 +1.057664 +1.077144 +0.106136 +0.936847 +1.000000 -0.879198 +0.897193 +1.139292
-0.006410 +1.143261 +2.103488 +0.043674 +1.828102 +344.000000 +258.000000 1 1 2 3 4 5 6 9 7 8 10 12
18 11 13 14 15 16 19 17 20 -1.681993 +0.043305 -1.730142 -1.398684 -0.000854 -1.399105 -1.138636
-0.010206 -1.146848 -0.975206 -0.131459 -0.845286 -0.866027 +0.655896 -0.758113 -0.737511 -0.088489
-0.737805 -0.572698 -0.011430 -0.545475 -0.515603 +0.011552 -0.535448 -0.577730 -0.198302 -0.196329
+0.129699 -0.010305 +0.122504 +0.377807 +0.013654 +0.183586 +0.180395 +0.093112 +0.634700 +0.417487
-0.058592 +0.538337 +0.611155 -0.025135 +0.607146 +0.879095 +0.016780 +0.849553 +0.987460 -0.045751
+1.057664 +1.181326 +0.104182 +0.936847 +0.350229 -0.649771 +0.897193 +1.140177 +0.000885 +1.143261
+1.719644 -0.383845 +1.828102 +358.000000 +258.000000 1 1 2 3 4 6 7 8 9 5 10 11 13 14 18 15 17 12 16
19 20 -1.729006 -0.047013 -1.730142 -1.398261 +0.000423 -1.399105 -1.153103 -0.014468 -1.146848
-1.107175 -0.131968 -0.845286 -0.355767 +0.510260 -0.758113 -0.796078 -0.058567 -0.737805 -0.599741
-0.027042 -0.545475 -0.521596 -0.005992 -0.535448 -0.431856 +0.145874 -0.196329 +0.101816 -0.027883
+0.122504 +0.278289 -0.099518 +0.183586 +0.860081 +0.679686 +0.634700 +0.486449 +0.068962 +0.538337
+0.578870 -0.032285 +0.607146 +0.831094 -0.048001 +0.849553 +1.060995 +0.073535 +1.057664 +0.842302
-0.339024 +0.936847 +0.687111 +0.336882 +0.897193 +1.147023 +0.006846 +1.143261 +1.636307 -0.083337
+1.828102 +304.000000 +258.000000 1 1 2 5 3 4 6 7 8 9 11 10 12 13 14 17 15 16 19 18 20 -1.741033
-0.012028 -1.730142 -1.397537 +0.000724 -1.399105 -1.148234 +0.004869 -1.146848 -1.118189 -0.011014
-0.845286 -1.355767 -1.000000 -0.758113 -0.730198 +0.065880 -0.737805 -0.550424 +0.049317 -0.545475
-0.524758 -0.003162 -0.535448 -0.081399 +0.350456 -0.196329 +0.108294 +0.006478 +0.122504 +0.065625
-0.212663 +0.183586 +0.518505 -0.341576 +0.634700 +0.605618 +0.119169 +0.538337 +0.620174 +0.041303
+0.607146 +0.824751 -0.006343 +0.849553 +1.034894 -0.026101 +1.057664 +0.819665 -0.022636 +0.936847
+1.508042 +0.820931 +0.897193 +1.145682 -0.001341 +1.143261 +1.826967 +0.190660 +1.828102
+302.000000 +258.000000 1 1 2 5 3 4 6 7 8 9 11 10 12 13 14 15 16 19 17 18 20 -1.674210 +0.066823
-1.730142 -1.397764 -0.000226 -1.399105 -1.118061 +0.030173 -1.146848 -1.054044 +0.064145 -0.845286
-1.311551 +0.044216 -0.758113 -0.647232 +0.082966 -0.737805 -0.584850 -0.034426 -0.545475 -0.498660
+0.026098 -0.535448 -0.047448 +0.033951 -0.196329 +0.138477 +0.030184 +0.122504 +0.094753 +0.029127
+0.183586 +0.265279 -0.253226 +0.634700 +0.610180 +0.004561 +0.538337 +0.635423 +0.015249 +0.607146
+0.834878 +0.010128 +0.849553 +1.035417 +0.000523 +1.057664 +1.255422 +0.435757 +0.936847 +1.432702
-0.075340 +0.897193 +1.141872 -0.003810 +1.143261 +1.987916 +0.160949 +1.828102 +302.000000
+258.000000 1 1 2 3 4 6 7 8 9 5 10 11 12 13 14 15 18 16 17 19 20 -1.670272 +0.003938 -1.730142
-1.398175 -0.000412 -1.399105 -1.134687 -0.016626 -1.146848 -1.018875 +0.035169 -0.845286 -0.311551
+1.000000 -0.758113 -0.659437 -0.012206 -0.737805 -0.602811 -0.017961 -0.545475 -0.527191 -0.028532
-0.535448 -0.493471 -0.446022 -0.196329 +0.125848 -0.012630 +0.122504 +0.379254 +0.284502 +0.183586
+0.456398 +0.191120 +0.634700 +0.517220 -0.092959 +0.538337 +0.609403 -0.026020 +0.607146 +0.865752
+0.030873 +0.849553 +1.001148 -0.034268 +1.057664 +1.020529 -0.234893 +0.936847 +0.950665 -0.482038
+0.897193 +1.140826 -0.001046 +1.143261 +2.070503 +0.082586 +1.828102 +272.000000 +258.000000 1 1 2
3 4 6 9 5 7 8 10 13 11 12 14 18 17 15 16 19 20 -1.765263 -0.094991 -1.730142 -1.398810 -0.000635
```

```
-1.399105 -1.154452 -0.019765 -1.146848 -0.948260 +0.070615 -0.845286 -0.653286 -0.341735 -0.758113
-0.758483 -0.099046 -0.737805 -0.576729 +0.026082 -0.545475 -0.545309 -0.018117 -0.535448 -0.715008
-0.221537 -0.196329 +0.104546 -0.021302 +0.122504 +0.381197 +0.001943 +0.183586 +0.569630 +0.113232
+0.634700 +0.269820 -0.247400 +0.538337 +0.576565 -0.032838 +0.607146 +0.869696 +0.003945 +0.849553
+0.987605 -0.013544 +1.057664 +0.835017 -0.185512 +0.936847 +0.658326 -0.292339 +0.897193 +1.142577
+0.001750 +1.143261 +2.079209 +0.008707 +1.828102 +322.000000 +258.000000 1 1 2 5 3 4 6 7 8 9 11 10
13 12 14 15 17 18 16 19 20 -1.714510 +0.050752 -1.730142 -1.397682 +0.001128 -1.399105 -1.157918
-0.003466 -1.146848 -0.892069 +0.056190 -0.845286 -1.169339 -0.516053 -0.758113 -0.758146 +0.000337
-0.737805 -0.553743 +0.022986 -0.545475 -0.491803 +0.053506 -0.535448 -0.373501 +0.341507 -0.196329
+0.111998 +0.007452 +0.122504 +0.083330 -0.297867 +0.183586 +0.437196 -0.132434 +0.634700 +0.402998
+0.133178 +0.538337 +0.591160 +0.014595 +0.607146 +0.839854 -0.029842 +0.849553 +1.068460 +0.080856
+1.057664 +0.873573 +0.038556 +0.936847 +1.000152 +0.341826 +0.897193 +1.144764 +0.002187 +1.143261
+1.863074 -0.216135 +1.828102 +302.000000 +258.000000 1 1 2 3 4 5 6 7 8 11 9 12 10 13 14 15 16 19 18
17 20 -1.650246 +0.064265 -1.730142 -1.397376 +0.000306 -1.399105 -1.117832 +0.040086 -1.146848
-0.932163 -0.040093 -0.845286 -0.754135 +0.415204 -0.758113 -0.685045 +0.073102 -0.737805 -0.595466
-0.041723 -0.545475 -0.491104 +0.000698 -0.535448 -0.027594 +0.345907 -0.196329 +0.132521 +0.020523
+0.122504 -0.059945 -0.143276 +0.183586 +0.129617 -0.307580 +0.634700 +0.528656 +0.125658 +0.538337
+0.628031 +0.036871 +0.607146 +0.842760 +0.002906 +0.849553 +1.050853 -0.017607 +1.057664 +1.269995
+0.396422 +0.936847 +1.191260 +0.191108 +0.897193 +1.142835 -0.001928 +1.143261 +1.747849 -0.115225
+1.828102 +278.000000 +258.000000 1 1 2 3 4 5 6 7 8 9 11 10 12 13 14 15 19 16 18 17 20 -1.756130
-0.105885 -1.730142 -1.397946 -0.000570 -1.399105 -1.132315 -0.014483 -1.146848 -1.066229 -0.134067
-0.845286 -0.713203 +0.040932 -0.758113 -0.662261 +0.022784 -0.737805 -0.591179 +0.004287 -0.545475
-0.561377 -0.070273 -0.535448 -0.226702 -0.199108 -0.196329 +0.125411 -0.007110 +0.122504 +0.108434
+0.168379 +0.183586 +0.306265 +0.176648 +0.634700 +0.524708 -0.003949 +0.538337 +0.623058 -0.004972
+0.607146 +0.859993 +0.017233 +0.849553 +1.028080 -0.022773 +1.057664 +1.261055 -0.008940 +0.936847
+1.060325 -0.130936 +0.897193 +1.000000 -0.083840 +1.143261 +2.075321 +0.327472 +1.828102
+272.000000 +258.000000 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 16 19 20 -1.821176 -0.065046
-1.730142 -1.399674 -0.001728 -1.399105 -1.150848 -0.018533 -1.146848 -0.974914 +0.091315 -0.845286
-0.812999 -0.099795 -0.758113 -0.688640 -0.026379 -0.737805 -0.539415 +0.051764 -0.545475 -0.510408
+0.050969 -0.535448 -0.402091 -0.175389 -0.196329 +0.119627 -0.005784 +0.122504 +0.315959 +0.207525
+0.183586 +0.451767 +0.145503 +0.634700 +0.513384 -0.011323 +0.538337 +0.576473 -0.046585 +0.607146
+0.855716 -0.004277 +0.849553 +1.046667 +0.018587 +1.057664 +0.875164 -0.385890 +0.936847 +0.917132
-0.143193 +0.897193 +1.230747 +0.230747 +1.143261 +1.989392 -0.085929 +1.828102 +240.000000
+258.000000 1 1 2 3 5 4 6 7 9 8 10 13 12 11 14 17 15 18 16 19 20 -1.839620 -0.018444 -1.821176
-1.399554 +0.000120 -1.399674 -1.147411 +0.003437 -1.150848 -0.929147 +0.045767 -0.974914 -1.038503
-0.225504 -0.812999 -0.686714 +0.001925 -0.688640 -0.526619 +0.012795 -0.539415 -0.468819 +0.041589
-0.510408 -0.525361 -0.123270 -0.402091 +0.106415 -0.013212 +0.119627 +0.446018 +0.130058 +0.315959
+0.340789 -0.110978 +0.451767 +0.319901 -0.193484 +0.513384 +0.554515 -0.021958 +0.576473 +0.852635
-0.003081 +0.855716 +1.002243 -0.044425 +1.046667 +0.687538 -0.187626 +0.875164 +0.958850 +0.041718
+0.917132 +1.283180 +0.052433 +1.230747 +1.956853 -0.032539 +1.989392 +284.000000 +240.000000 1 1 2
3 4 5 6 7 8 9 10 12 13 11 14 17 15 16 18 19 20 -1.610671 +0.228949 -1.821176 -1.398181 +0.001373
-1.399674 -1.129495 +0.017916 -1.150848 -0.965397 -0.036249 -0.974914 -0.965271 +0.073232 -0.812999
-0.664133 +0.022581 -0.688640 -0.586668 -0.060049 -0.539415 -0.494832 -0.026013 -0.510408 -0.338385
+0.186976 -0.402091 +0.114876 +0.008461 +0.119627 +0.334005 -0.112012 +0.315959 +0.328117 -0.012672
+0.451767 +0.333989 +0.014088 +0.513384 +0.622928 +0.068413 +0.576473 +0.854857 +0.002222 +0.855716
+0.980118 -0.022125 +1.046667 +0.723815 +0.036277 +0.875164 +1.013213 +0.054363 +0.917132 +1.056736
-0.226444 +1.230747 +1.981214 +0.024361 +1.989392 +262.000000 +240.000000 1 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 -1.698265 -0.087595 -1.821176 -1.397663 +0.000519 -1.399674 -1.143139
-0.013644 -1.150848 -1.029168 -0.063772 -0.974914 -0.791770 +0.173501 -0.812999 -0.655317 +0.008816
-0.688640 -0.610774 -0.024106 -0.539415 -0.523429 -0.028597 -0.510408 -0.325858 +0.012527 -0.402091
```

```
+0.119564 +0.004688 +0.119627 +0.213655 -0.120351 +0.315959 +0.329930 +0.001814 +0.451767 +0.529900
+0.195911 +0.513384 +0.645808 +0.022879 +0.576473 +0.857603 +0.002746 +0.855716 +1.002584 +0.022467
+1.046667 +1.008310 +0.284494 +0.875164 +1.057324 +0.044111 +0.917132 +1.123406 +0.066669 +1.230747
+2.008291 +0.027077 +1.989392 +230.000000 +240.000000 1
```

Appendix D

Experiment Task Sets

This appendix contains a collection of the tasks sets used throughout the load (Sec. 6.3) and scalability (Sec. 6.4) test. The data used to generate these plots is available in the accompanied CD under the *output* folder as described in Appendix E.

D.1 Load Experiment

D.1.1 Experiment: $C_N = 1$, Load 10%

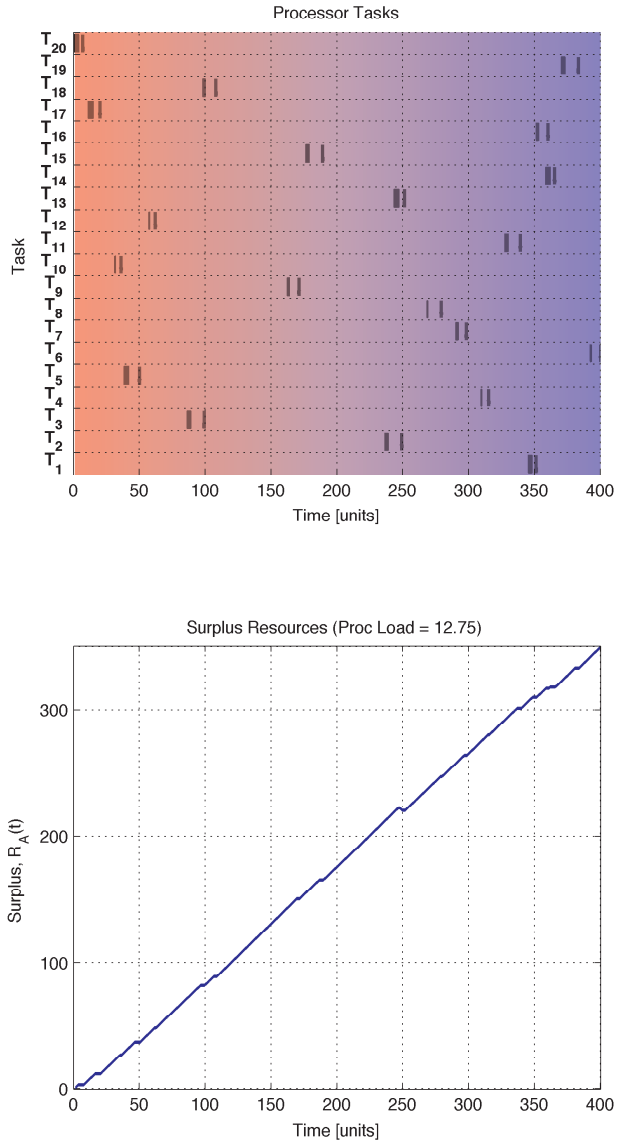


Fig. D.1: Task definition for $C_N = 1$ with 10% system load (/test02-p1-t20-load10). The processor load estimate assumes a single core.

D.1.2 Experiment: $C_N = 1$, Load 50%

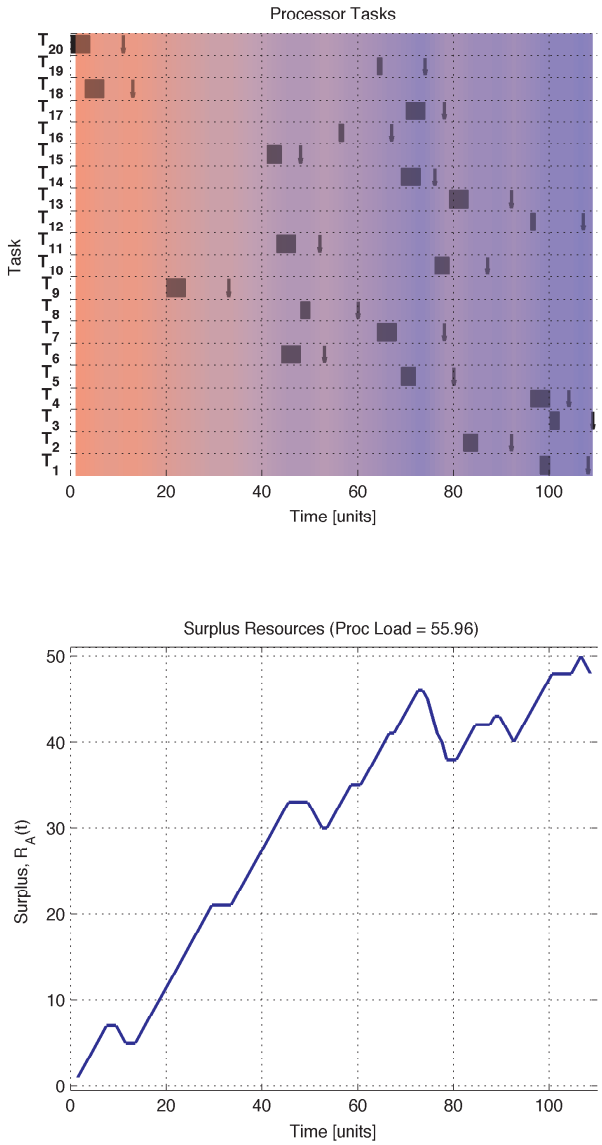


Fig. D.2: Task definition for $C_N = 1$ with 50% system load (/test03-p1-t20-load50). The processor load estimate assumes a single core.

D.1.3 Experiment: $C_N = 1$, Load 80%

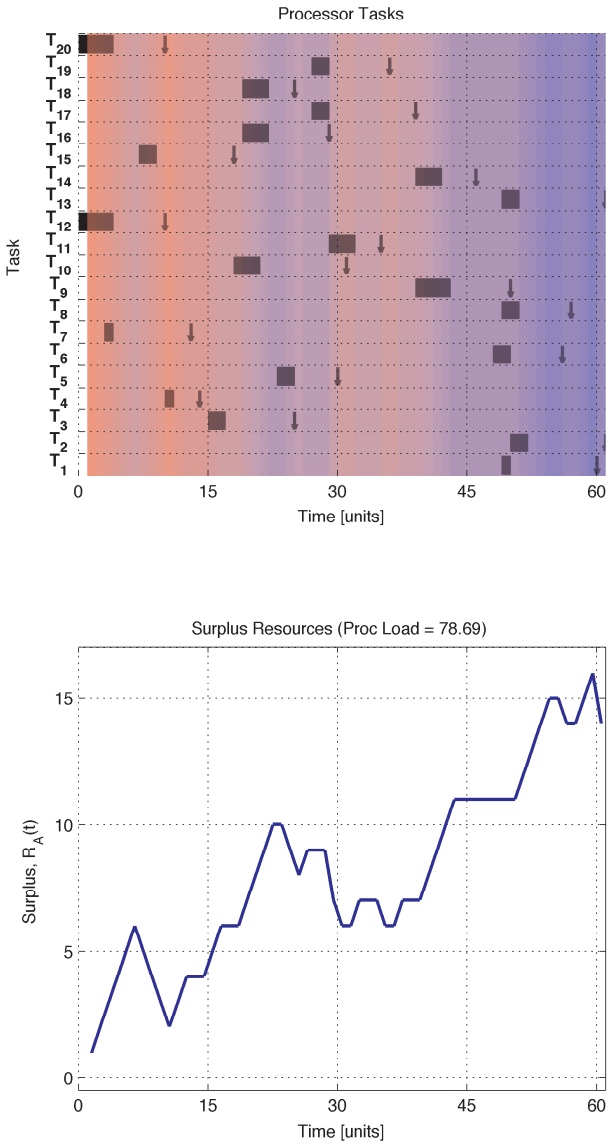


Fig. D.3: Task definition for $C_N = 1$ with 80% system load (/test04-p1-t20-load80). The processor load estimate assumes a single core.

D.1.4 Experiment: $C_N = 2$, Load 10%

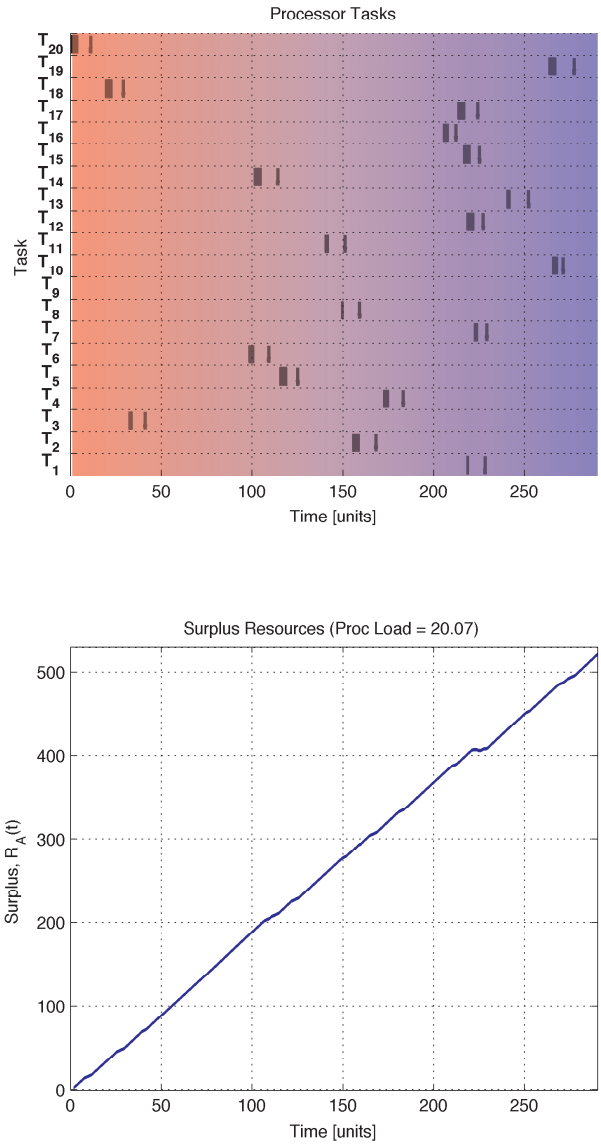


Fig. D.4: Task definition for $C_N = 2$ with 10% system load (/test06-p2-t20-load10). The processor load estimate assumes a single core.

D.1.5 Experiment: $C_N = 2$, Load 50%

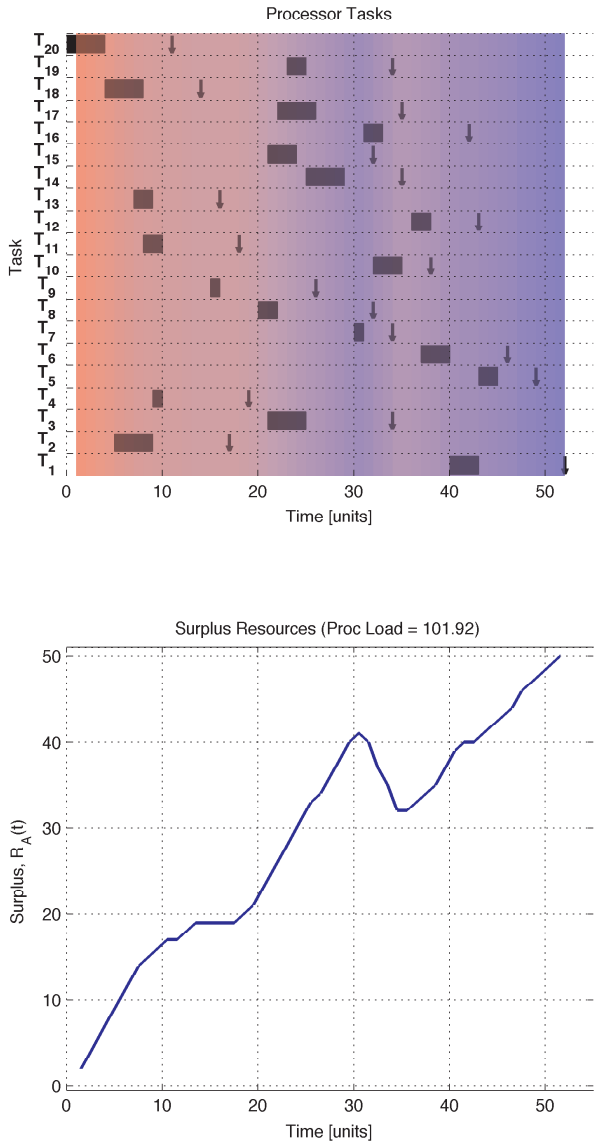


Fig. D.5: Task definition for $C_N = 2$ with 50% system load (/test07-p2-t20-load50). The processor load estimate assumes a single core.

D.1.6 Experiment: $C_N = 2$, Load 80%

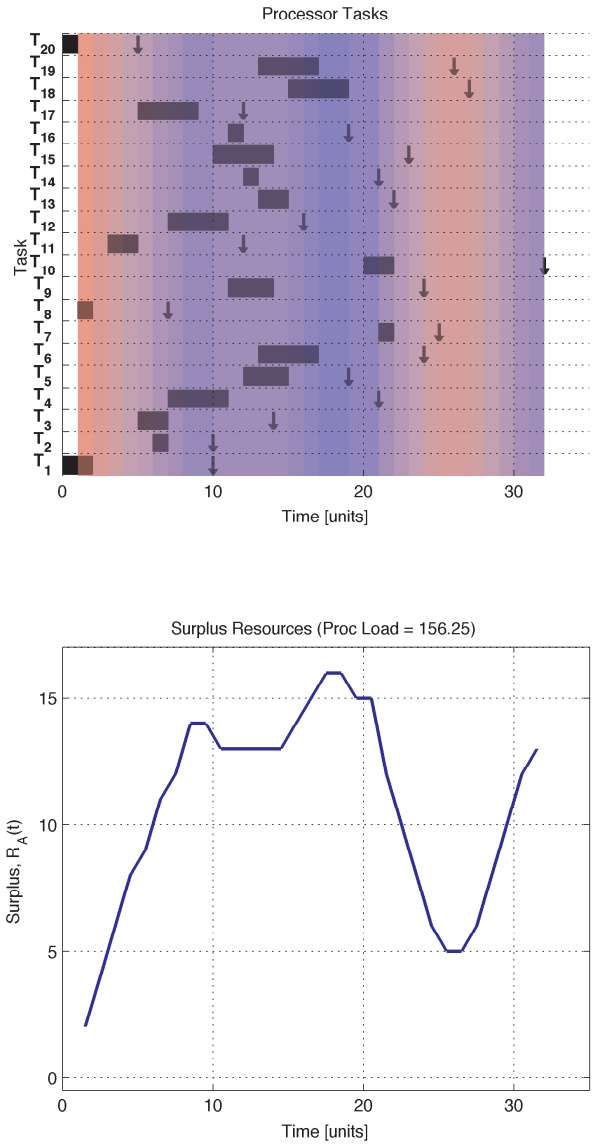


Fig. D.6: Task definition for $C_N = 2$ with 80% system load (/test08-p2-t20-load80). The processor load estimate assumes a single core.

D.1.7 Experiment: $C_N = 4$, Load 10%

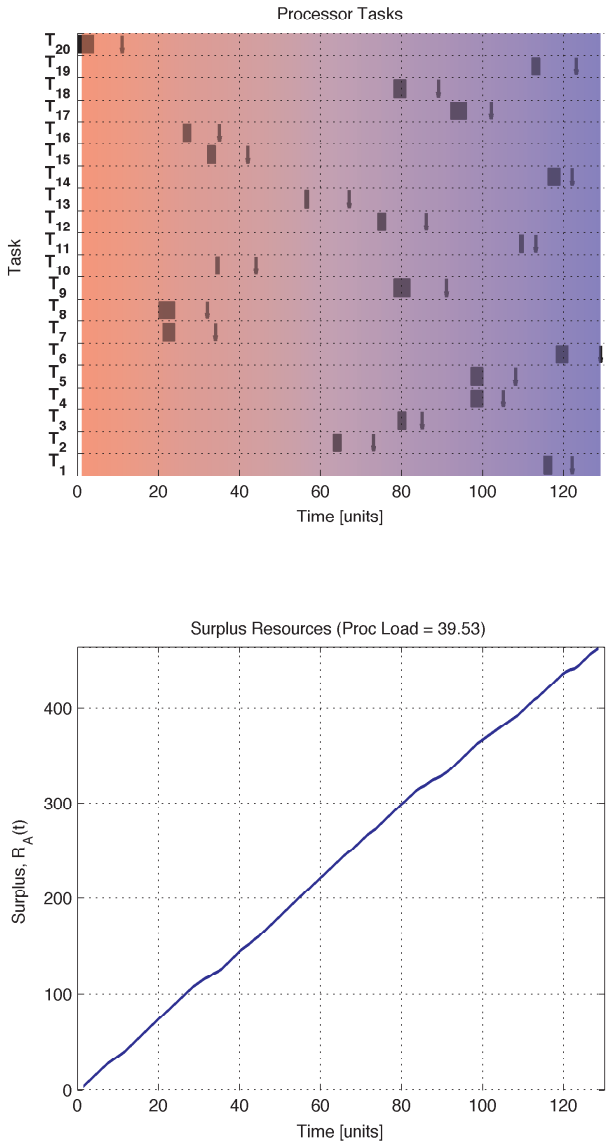


Fig. D.7: Task definition for $C_N = 4$ with 10% system load (/test10-p4-t20-load10). The processor load estimate assumes a single core.

D.1.8 Experiment: $C_N = 4$, Load 50%

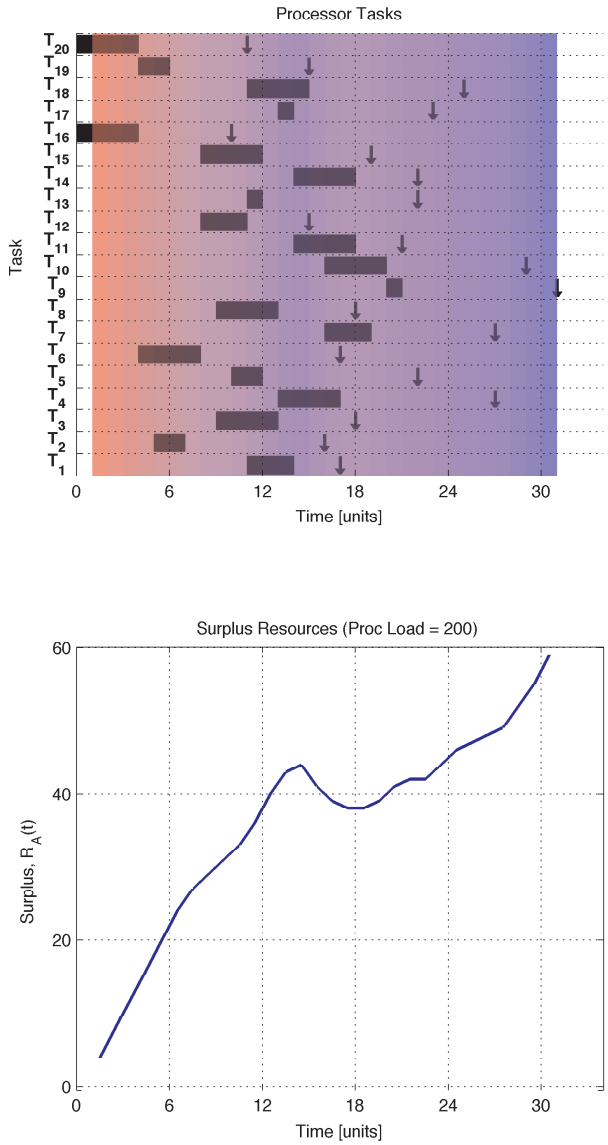


Fig. D.8: Task definition for $C_N = 4$ with 50% system load (/test11-p4-t20-load50). The processor load estimate assumes a single core.

D.1.9 Experiment: $C_N = 4$, Load 80%

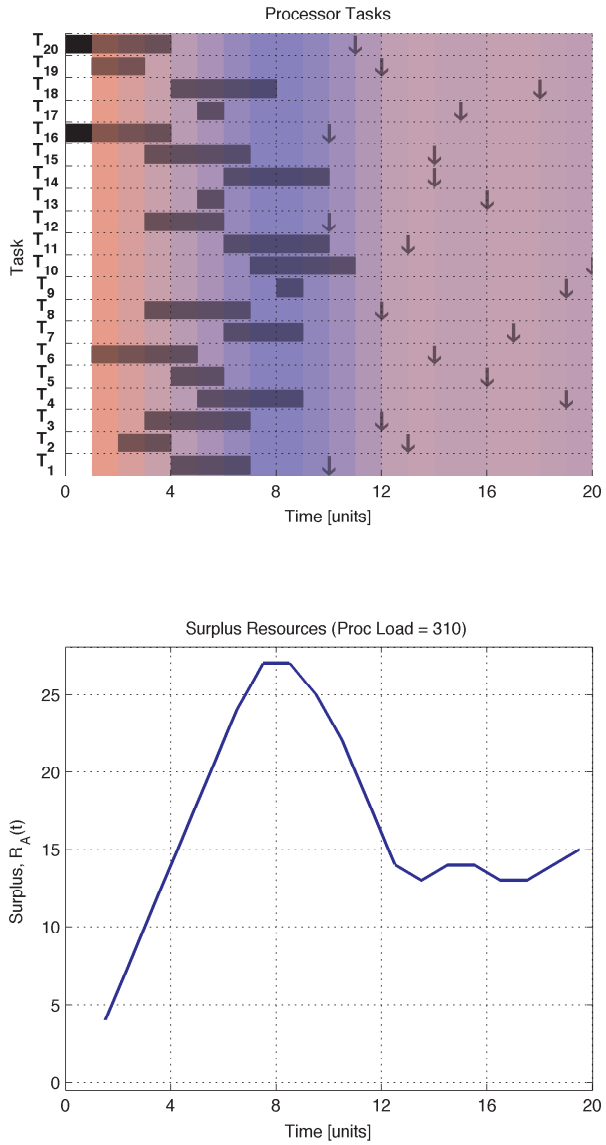


Fig. D.9: Task definition for $C_N = 4$ with 80% system load (/test12-p4-t20-load80). The processor load estimate assumes a single core.

D.2 Scalability Experiment

D.2.1 Experiment: $C_N = 4$, Load 50%, $T_N = 20$

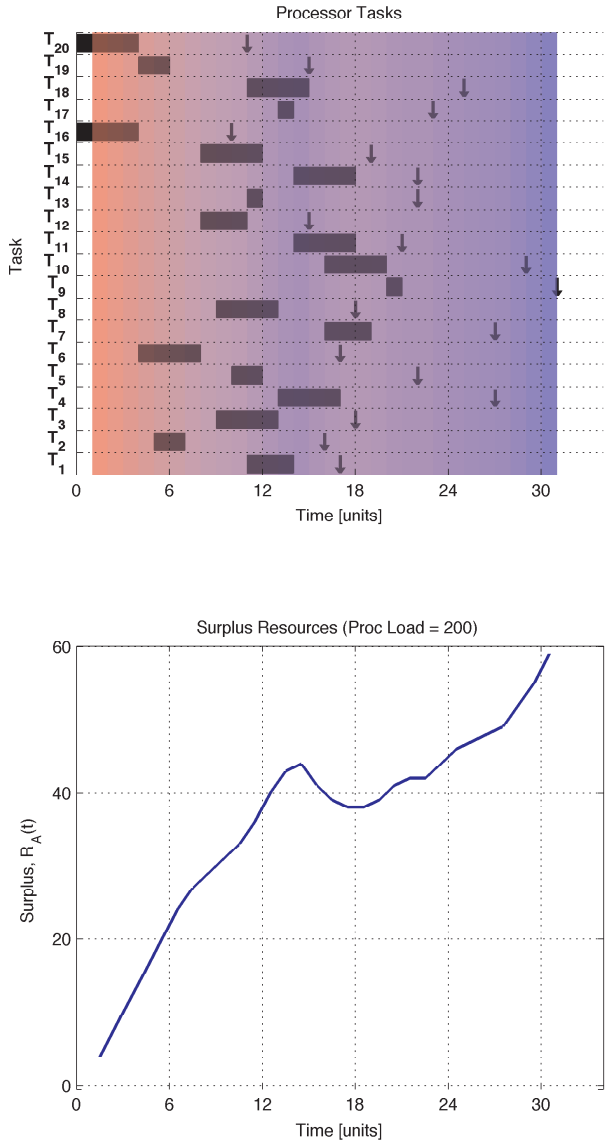


Fig. D.10: Task definition for $C_N = 4$ with 50% system load, and 20 tasks (/test11-p4-t20-load50). The processor load estimate assumes a single core.

D.2.2 Experiment: $C_N = 4$, Load 50%, $T_N = 30$

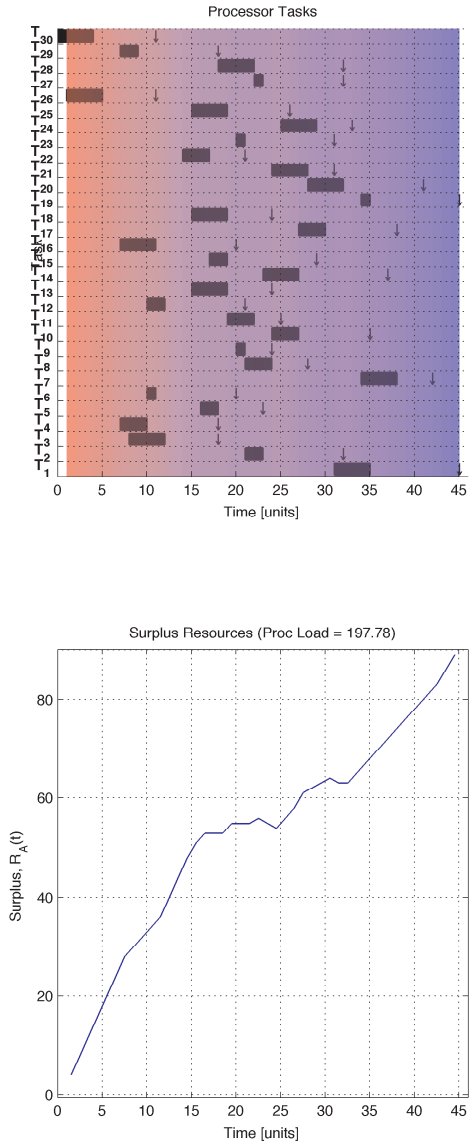


Fig. D.11: Task definition for $C_N = 4$ with 50% system load, and 30 tasks (/test13-p4-t30-load50). The processor load estimate assumes a single core.

D.2.3 Experiment: $C_N = 4$, Load 50%, $T_N = 40$

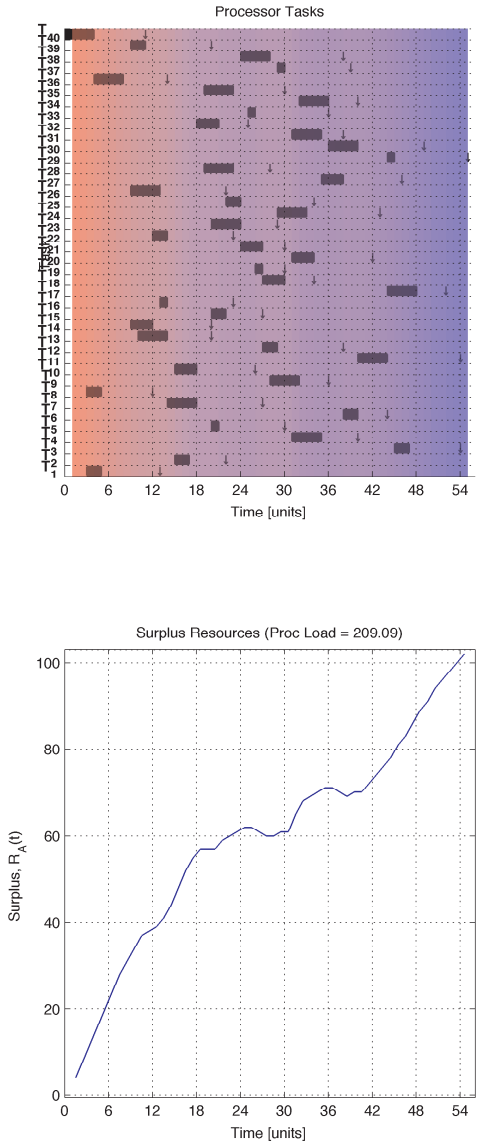


Fig. D.12: Task definition for $C_N = 4$ with 50% system load, and 40 tasks (/test14-p4-t40-load50). The processor load estimate assumes a single core.

D.2.4 Experiment: $C_N = 4$, Load 50%, $T_N = 50$

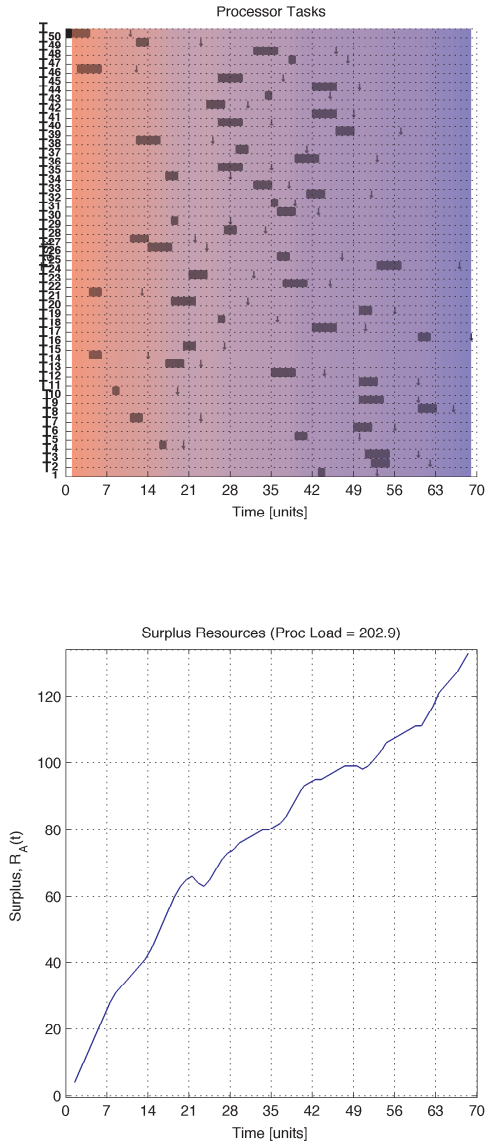


Fig. D.13: Task definition for $C_N = 4$ with 50% system load, and 50 tasks (/test15-p4-t50-load50). The processor load estimate assumes a single core.

Appendix E

DVD Contents

There are 4 DVDs accompanying this thesis. The source code is contained in Disc 1 under the *simulator* folder. The remaining zip files contain the full results of all the experiments described in Ch. 6.

E.1 Disc 1 Contents

```
+-- [287M]  griewank_vMax_0.2.zip
+-- [359M]  rastrigin_vMax_0.2.zip
+-- [365M]  rosenbrock_vMax_0.2.zip
+-- [ 714]  simulator
|   +-- [ 340]  TimeSeriesAnalysis
|   |   +-- [ 204]  Correlation
|   |   |   +-- [ 589]  ts_correlation.m
|   |   |   +-- [1.7K]  ts_correlation_tb.m
|   |   +-- [ 204]  CorrelationTrajectory
|   |   |   +-- [1023]  ts_corr_trajectory.m
|   |   |   +-- [ 943]  ts_corr_trajectory_tb.m
|   |   +-- [ 170]  Distribution
|   |   |   +-- [ 441]  ts_distribution.m
|   |   |   +-- [ 681]  ts_distribution_tb.m
|   |   +-- [ 170]  EnvelopeDetector
|   |   |   +-- [ 938]  sample1.m
|   |   +-- [ 204]  FFT
|   |   |   +-- [1.5K]  ts_powerspectrum.m
|   |   |   +-- [1.7K]  ts_powerspectrum_tb.m
|   |   +-- [ 204]  Moments
|   |       +-- [2.4K]  ts_moments.m
|   |       +-- [1.7K]  ts_moments_tb.m
```

```
|   +-- [ 340]  etc
|   |   +-- [3.4K]  applyhatch.m
|   |   +-- [6.6K]  drawaxis.m
|   |   +-- [ 30K]  exportfig.m
|   |   +-- [1.5K]  license.txt
|   |   +-- [ 943]  makehatch.m
|   |   +-- [ 562]  print_same_line.m
|   |   +-- [1.6K]  print_time_left.m
|   |   +-- [ 872]  progress_bar.m
|   +-- [  69]  initializeTest.m
|   +-- [4.8K]  main_sched.m
|   +-- [3.7K]  main_test.m
|   +-- [ 578]  pso
|   |   +-- [ 408]  functions
|   |   |   +-- [ 30K]  exportfig.m
|   |   |   +-- [ 290]  griewank.m
|   |   |   +-- [1.7K]  initializeFunction.m
|   |   |   +-- [1007]  main.m
|   |   |   +-- [ 214]  rastrigin.m
|   |   |   +-- [ 233]  rosenbrock.m
|   |   |   +-- [ 191]  sphere.m
|   |   |   +-- [ 671]  test_function.m
|   |   +-- [1.4K]  initializePlot.m
|   |   +-- [ 789]  main2.m
|   |   +-- [7.6K]  pso.m
|   |   +-- [4.6K]  psoND2.m
|   |   +-- [7.2K]  psoreg.m
|   |   +-- [2.1K]  rate1.m
|   |   +-- [ 478]  readEntries.m
|   |   +-- [ 777]  readEntries2.m
|   |   +-- [ 559]  readEntries3.m
|   |   +-- [ 374]  topologies
|   |   |   +-- [ 367]  gBest.m
|   |   |   +-- [1.5K]  initializeTopology.m
|   |   |   +-- [1.5K]  lBestGov.m
|   |   |   +-- [ 828]  lBestGrid.m
|   |   |   +-- [ 755]  lBestRing.m
|   |   |   +-- [ 444]  lBestStar.m
|   |   |   +-- [1.2K]  lBestTree.m
|   |   +-- [1.3K]  trajectory3D.m
|   |   +-- [1.2K]  updatePlot.m
|   |   +-- [ 238]  weights
|   |       +-- [ 330]  initializeSocialWeights.m
|   |       +-- [ 568]  phiRandom.m
|   |       +-- [  89]  phiShEb99.m
|   |       +-- [  81]  phiTrel02.m
|   +-- [1.2K]  read_struct.m
|   +-- [1.9K]  save_struct.m
|   +-- [ 612]  sched
|   |   +-- [2.4K]  colorGradient.m
```

```
|   |   +-- [ 339]  create_task.m
|   |   +-- [ 204]  distribution
|   |   |   +-- [ 104]  normal.m
|   |   |   +-- [  88]  poisson.m
|   |   |   +-- [ 650]  test_distribution.m
|   |   |   +-- [ 126]  uniform.m
|   |   +-- [1.6K]  gen_tasks.m
|   |   +-- [1.6K]  gen_tasks2.m
|   |   +-- [ 803]  is_schedulable.m
|   |   +-- [ 910]  minTotalTardiness.m
|   |   +-- [1.8K]  minTotalTardinessN.m
|   |   +-- [1.8K]  minTotalTardinessNplot.m
|   |   +-- [ 279]  plot_deadline.m
|   |   +-- [1.4K]  plot_game.m
|   |   +-- [1.9K]  plot_sched.m
|   |   +-- [3.4K]  plot_task_timing.m
|   |   +-- [ 351]  read_tasks.m
|   |   +-- [ 339]  save_tasks.m
|   +-- [ 136]  sims
|   |   +-- [ 170]  sample
|   |       +-- [1.1K]  config.txt
|   |       +-- [ 338]  tasks.txt
|   +-- [1.4K]  test_cost_function.m
|   +-- [ 775]  test_plot.m
|   +-- [ 713]  test_save.txt
|   +-- [ 845]  test_sched.m
|   +-- [ 557]  test_sched_cost.m
|   +-- [ 264]  testw.m
|   +-- [ 850]  thesis-figure
|   |   +-- [ 777]  f02_s01_taskdef.m
|   |   +-- [1.7K]  f02_s02_gantt.m
|   |   +-- [2.7K]  f02_s03_game.m
|   |   +-- [2.7K]  f02_s04_game.m
|   |   +-- [1.8K]  f04_s12_encoding.m
|   |   +-- [1.8K]  f04_s13_encoding.m
|   |   +-- [3.2K]  f04_s14_cosftfunction.m
|   |   +-- [2.2K]  f05_s01_taskdef.m
|   |   +-- [1.7K]  f06_s01_benchmark.m
|   |   +-- [1.9K]  f06_s02_timeseries.m
|   |   +-- [1.6K]  f06_s03_taskdef.m
|   |   +-- [1.6K]  f06_s03_taskdef.m~
|   |   +-- [1.8K]  f06_s04_sched_clean.m
|   |   +-- [1.9K]  f06_s05_timeseries.m
|   |   +-- [1.5K]  f06_s06_correlation.m
|   |   +-- [1.5K]  f06_s06_moments.m
|   |   +-- [1.5K]  f06_s06_spectrum.m
|   |   +-- [2.8K]  plot_cost_function.m
|   |   +-- [1.8K]  plot_cost_function2.m
|   |   +-- [2.4K]  plot_timeseries.m
|   |   +-- [ 272]  tasks.txt
```

```
|   |   +-- [ 336]  tasks2.txt
|   +-- [1013]  untitled.m
+-- [341M]  sphere_vMax_0.2.zip
+-- [100M]  test01-p1-t12-benchmark.zip
+-- [ 71M]  test02-p1-t20-load10.zip
+-- [121M]  test03-p1-t20-load50.zip
+-- [ 36M]  test04-p1-t20-load80.zip
+-- [316M]  test05-p2-t12-benchmark.zip
+-- [121M]  test06-p2-t20-load10.zip
+-- [ 48M]  test07-p2-t20-load50.zip
```

E.2 Disc 2 Contents

```
+-- [1.6G]  test08-p2-t20-load80.zip
+-- [ 86M]  test09-p4-t12-benchmark.zip
+-- [109M]  test10-p4-t20-load10.zip
+-- [ 83M]  test11-p4-t20-load50.zip
+-- [555M]  test12-p4-t20-load80.zip
+-- [1.1G]  test13-p4-t30-load50.zip
```

E.3 Disc 3 Contents

```
+-- [2.8G]  test14-p4-t40-load50.zip
```

E.4 Disc 4 Contents

```
+-- [3.7G]  test15-p4-t50-load50.zip
```

Appendix F

Colophon

This thesis is typeset in LaTeX using a custom template under TeXShop Version 3.11 that utilizes the TeXLive-2012 engine. The references are managed using BibTeX version 0.99d. The body is written in 11 point Times New Roman with figure captions displayed in 10 point Arial.

Figures 2.4 and 4.18 are created in Microsoft Visio 2010 Professional. All other figures are generated using Matlab version 2010b and saved as Encapsulated PostScripts (eps) files using the `exportfig` script written by Ben Hinkle and available online at the Matlab FileExchange Central (http://www.mathworks.com/matlabcentral/fileexchange/727-exportfig).

The majority of the work was performed using a Mac OS X version 10.6.8 and Windows 7 Professional running as dual boots in a 2×2.4 GHz Quad-Core Intel Xeon processors with 12 GB of 1066 MHz DDR3 memory. Some writing and editing was completed on Mac OS X version 10.8.3 running on a 3.6 GHz Intel Core i5 with 4 GB of 1333 MHz RAM.

The source code and LaTeX files are managed using a secure Subversion repository hosted at http://www.darioschor.com.