University of the Witwatersrand, Johannesburg

A new hybrid meta-heuristic algorithm for solving single machine scheduling problems

by

Natasha Zlobinsky

A dissertation submitted in partial fulfilment of the
degree of Master of Science in Engineering (Electrical) (50/50)
in the

Faculty of Engineering and the Built Environment
Department of Electrical and Information Engineering

May 2017

Declaration of Authorship

I, Natasha Zlobinsky, declare that this dissertation titled, 'A new hybrid meta-heuristic algorithm for solving single machine scheduling problems' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: [illegible]

Date: 23/05/2017

“ - Nick Fury: How bad is it?
- Agent Phil Coulson: That’s the problem, sir. We don’t know”

The Avengers (2012)

“ Sometimes you have to go through something else to find what you’re looking for”

Robert Genn

“One does not stand still looking for a path. One walks; and as one walks, a path comes into being”

Mas Kodani

Abstract

Numerous applications in a wide variety of fields has resulted in a rich history of research into optimisation for scheduling. Although it is a fundamental form of the problem, the single machine scheduling problem with two or more objectives is known to be NP-hard. For this reason we consider the single machine problem a good test bed for solution algorithms. While there is a plethora of research into various aspects of scheduling problems, little has been done in evaluating the performance of the Simulated Annealing algorithm for the fundamental problem, or using it in combination with other techniques. Specifically, this has not been done for minimising total weighted earliness and tardiness, which is the optimisation objective of this work.

If we consider a mere ten jobs for scheduling, this results in over 3.6 million possible solution schedules. It is thus of definite practical necessity to reduce the search space in order to find an optimal or acceptable suboptimal solution in a shorter time, especially when scaling up the problem size. This is of particular importance in the application area of packet scheduling in wireless communications networks where the tolerance for computational delays is very low. The main contribution of this work is to investigate the hypothesis that inserting a step of pre-sampling by Markov Chain Monte Carlo methods before running the Simulated Annealing algorithm on the pruned search space can result in overall reduced running times.

The search space is divided into a number of sections and Metropolis-Hastings Markov Chain Monte Carlo is performed over the sections in order to reduce the search space for Simulated Annealing by a factor of 20 to 100. Trade-offs are found between the run time and number of sections of the pre-sampling algorithm, and the run time of Simulated Annealing for minimising the percentage deviation of the final result from the optimal solution cost. Algorithm performance is determined both by computational complexity and the quality of the solution (i.e. the percentage deviation from the optimal). We find that the running time can be reduced by a factor of 4.5 to ensure a 2% deviation from the optimal, as compared to the basic Simulated Annealing algorithm on the full search space. More importantly, we are able to reduce the complexity of finding the optimal from $O(n.n!)$ for a complete search to $O(nN_S)$ for Simulated Annealing to $O(n(N_M r + N_S) + m)$ for the input variables n jobs, N_S SA iterations, N_M Metropolis-Hastings iterations, r inner samples and m sections.

Acknowledgements

I have been blessed to have some very patient and understanding people on my side during the course of this research. First and foremost, a huge vote of thanks to my amazing supervisor, Prof. Ling Cheng. I sincerely thank you for your assistance, direction, patience, understanding and invaluable contribution of ideas. Thank you for sticking with me despite my moving to another city, changing jobs and taking a year longer to complete the dissertation than I had originally planned! I would not have been able to finish this without you. Also to Ling's other students in the CeTAS group: Ayokunle (Fami), Ryan, Yves and Ashton, thank you for your camaraderie and sharing of ideas. I would also like to say that I appreciate Wits being understanding of my situation and allowing me the extensions necessitated by changing circumstances.

My thanks go to Dr. David Johnson for his patience and understanding, allowing me the time to complete this degree while starting the next and a new job. You are without exaggeration the best boss I have ever had.

To Richard Maliwatu, I am so grateful for your time in assisting me with the gargantuan task of moving this dissertation to LaTeX, as well as for your valuable advice and technical and emotional support. Thanks go to other lenders of emotional support and words of reassurance that there would indeed be a time when this would be complete: Janine Silberbauer, Danielle Retief and Lindsay Donaldson. Your friendship means more to me than you know! I also would like to extend my gratitude to Estie Boshoff and co. who jumped to my aid when the unimaginable happened and my laptop crashed along with two weeks of work. Mpendulo Ndlovu, my friend, thank you for your words of wisdom from one who has been here and done this, and reminding me not to take myself so seriously.

Mamma, dankie dat jy hierdeur gelees het en so baie foute opgetel het sodat ek nie 'n gek van myself gemaak het nie! Dankie vir alles wat jy nog oor die laaste drie jaar vir my gedoen het om my te help om hierdie graad klaar te maak, en om my lewe bietjie makliker te maak.

Thank you to any and all not mentioned by name for the countless votes of confidence and help both tangible and intangible.

Lastly, thank you to my examiner for giving this work an (albeit captive) audience.

Contents

List of Figures

List of Tables

Abbreviations

B&B	**B**ranch and **B**ound
CPU	**C**entral **P**rocessing **U**nit
CTS	**C**lear **t**o **S**end
DP	**D**ynamic **P**rogramming
EDD	**E**arliest **D**ue **D**ate
GA	**G**enetic **A**lgorithm
IEEE	**I**nstitute of **E**lectrical and **E**lectronic **E**ngineers
LCL	**L**east **C**ost **L**ast
LTE	**L**ong **T**erm **E**volution
MAC	**M**edium **A**ccess **C**ontrol (layer)
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MDP	**M**arkov **D**ecision **P**rocess
MH	**M**etropolis-**H**astings
NDTM	**N**on-**d**eterministic **T**uring **M**achine
NP	**N**on-deterministic **P**olynomial
NSE	**N**umeric **S**tandard **E**rror
p.d.f.	**p**robability **d**ensity **f**unction
PHY	**Phy**sical (layer)
QoS	**Q**uality **o**f **S**ervice
RDD	**R**elative **D**ue **D**ate
RTS	**R**equest **t**o **S**end
SA	**S**imulated **A**nnealing
SAM	**S**imulated **A**nnealing with **M**etropolis-Hastings sampling
SSDP	**S**uccessive **S**ublimation **D**ynamic **P**rogramming
SPT	**S**hortest **P**rocessing **T**ime

SRPT	**S**hortest **R**emaining **P**rocessing **T**ime
TS	**T**abu **S**earch
WSPT	**W**eighted **S**hortest **P**rocessing **T**ime

Symbols

J	set of jobs
j	index referring to a single job
p_j	processing time of job j
d_j	end of due window of job j
e_j	start of due window of job j
s_j	starting time of job j
C_j	completion time of job j
w'_j	earliness weight of job j
w''_j	tardiness weight of job j
E_j	earliness of job j
T_j	tardiness of job j
$\equiv$	identical to
$P(X) \equiv P_X$	probability distribution
$p(X = x) \equiv p(x)$	single probability value
$\mathbb{E}_{P_X}[f(x)]$	expectation of the function $f(x)$ over the probability density function P_X
$\gamma \equiv \sum\limits_{j \in J} (w'_j E_j + w''_j T_j)$	cost function to minimise
$\vee$	OR
$\wedge$	AND
$U(a, b)$	uniform distribution of real numbers between a and b
$\|S\|$	size of object or set S. If S is a set this is equal to the number of elements in S

Chapter 1

Introduction

1.1 Background

Scheduling can be described as the allocation of a set of tasks over any sparse resource in order to optimise certain objectives [1]. The applications are as numerous as the field of engineering is broad, making this an important area of research. With a history of research mainly centred on the manufacturing industry, traditional scheduling methods may come short of meeting the needs of more modern applications, where results may be more time-sensitive. In Telecommunications, for instance, tasks such as data packets may be cleverly scheduled in time or frequency domains, or both, in order to maximise data throughput. This needs to be done quickly enough to ensure the user experience is not negatively affected. Being typically an NP-hard or NP-complete problem – even with just one machine or processor, and a single optimisation objective – meta-heuristic algorithms are the accepted solution technique since no optimal polynomial time algorithm exists (unless $P = NP$) [2]. Finding the optimal solution is as easy as finding the proverbial needle in a haystack – without a magnet or other metal detector.

Some common algorithms used to date are Genetic Algorithms (GAs), Tabu Search (TS) and Simulated Annealing (SA), as well as Particle Swarm optimisation, to name but a few. These may provide acceptable answers in a much shorter time than optimal methods or the brute force approach, but there is still a time cost to pay. If a general way can be found to reduce the solution search space and reduce run times without eliminating potentially "good" solutions, this would be of great benefit to a variety of industries.

A few questions now emerge:

i) What algorithm to choose out of the ocean of possible options?

ii) Can any algorithm categorically be named the best?

iii) Can algorithms be combined to exploit advantages of each?

iv) Can performance be improved by adding pre-sampling to prune the search space?

It is not the aim of this work to answer all these questions but it does aim to shed some light on iv and to some extent iii. We have chosen the Simulated Annealing algorithm as the basis, partly motivated by an apparent shortage of research using this algorithm for the sum of weighted earliness and tardiness even though it appears to be an attractive option, but also owing to its comparative computational simplicity and shorter running time over the often-preferred Genetic Algorithm. Further justification of this choice can be found in Section 1.3.3 It is not by any means contended that this algorithm always displays better performance than others, and this is indeed irrelevant to the purpose of this work. Simply, the aim is to investigate the hypothesis that pre-sampling methods can be combined successfully with a basic SA algorithm in an advantageous way. A full motivation is presented in Section 1.3.

Although algorithms have become increasingly sophisticated and a number of options exist when a scheduling problem is to be solved, there is still a limit to the size of the problem for which we can find optimal solutions because the inherent computational complexity results in long running times. For scheduling problems this is of particular concern as the number of permutations (and thus the search space) grows very fast with the number of jobs. When fast results are required or the number of tasks is even as modest as ten, a method is required to reduce the time to obtain an answer of acceptable quality, as determined by the particular context. The question arises whether any gain can be had from combining techniques to trade off the strengths and weaknesses of each. SA is a fitting general algorithm to use when little is known of the problem and the search space does not follow any identifiable pattern or trend. The latter may seem true of scheduling problems in general but on closer investigation it may be seen that certain areas of the search space could be eliminated early in the search process to prevent the algorithm wasting time foraying into sections that statistically are unlikely to contain the optimal answer. Pre-sampling may offer a way to prune early without counterbalancing this improvement with more complexity.

In this work it is investigated if the performance of Simulated Annealing can be improved by combining the SA algorithm with other sampling methods in a two-step process. The focus thus turns to efficient sampling methods. Probably the most powerful sampling method available at this time is Markov Chain Monte Carlo (MCMC) methods based on Bayesian inference.

1.2 The Problem

The problem is to apply, or devise, and test an algorithm that is able to find solutions to single machine scheduling problems in a relatively short time period that are minimally deviant from the optimal value. Since the number of possible scheduling problem instances is infinite, our experiments are confined to a few instances only, which we contend are sufficiently representative for the results here to be of more general significance. Performance is evaluated in terms of the running time and quality of the solution: on the basis of analysis of the worst-case complexity, and percentage deviation from the optimum value, respectively. While we note that there are limitations to characterising run time by worst-case complexity analysis, a concern raised by Hall and Posner [3], this method is more fair for the purposes of a comparative study than measuring running times in seconds, since it enables the evaluation to be independent of implementation details and bias that would inevitably arise from the way a programmer writes the code for the different methods, or from platform differences (operating system, processor etc.).

We now list assumptions necessary to narrow the focus of the problem and provide a springboard for solutions.

Assumptions:

- The processing times of jobs are known *a priori* to the scheduling activity and all jobs are available at the start of the scheduling activity.
- Machine idle time is not allowed or required and no set-up time is considered.
- No pre-emption is allowed.
- There are no precedence relationships between jobs, so that any sequence of jobs is a feasible solution.
- The problem instances tested are sufficiently representative of real problem sets in as far as proving the methodology to be presented is valid and for relative performance evaluations to be at least indicative of expected results for any given problem instance.

The scheduling goal is to minimise the total weighted amount of time by which each job is completed outside of its due window, and we seek an efficient solution (i.e. one that satisfies the conditions given by eqn. (1.2)). This is an NP-hard problem [1, 4, 5]. Since machine idle time is considered unproductive and assumed unnecessary, it is not considered in this work; consequently the scheduling decision involves only the order in which jobs are to be scheduled so as to minimise the scheduling objective, eqn. (1.1)

$$min\gamma = min\sum_{j\in J}(w'_jE_j + w''_jT_j) \quad , \tag{1.1}$$

where jobs are given unique due windows (not just due dates) and earliness (E_j) and tardiness (T_j) carry different penalties per job, w'_j and w''_j, respectively. Please note that clarification and illustration of elements of scheduling are given in Section 2.1.

A schedule S is said to be efficient if there does not exist another schedule S_0 satisfying, over all $j \in J$,

$$\left\{\max_j[w'_jE_j(S_0)] < \max_j[w'_jE_j(S)]\right\} \wedge \left\{\max_j[w''_jT_j(S_0)] < \max_j[w''_jT_j(S)]\right\} \quad . \tag{1.2}$$

The null (H_0) and alternate (H_a) hypotheses investigated are:

- H_0: A combination of Simulated Annealing and Metropolis-Hastings Markov Chain Monte Carlo pre-sampling (to prune the search space) cannot be found that reduces the running time of SA (on the full search space) for the problem instances presented.
- H_a: A combination of Simulated Annealing and Metropolis-Hastings Markov Chain Monte Carlo pre-sampling (to prune the search space) can be found that reduces the running time of SA (on the full search space) for the problem instances presented.

To this point, we have evaded defining what a "reasonable", "acceptable" or "good" solution would be, since this is situation dependent and relative. Similarly, ambivalent terms for run times are used. For the problem instances investigated in this work, the issue has been circumvented by comparing results on a relative scale, and categorising solutions depending on their statistical properties. Detail on this is given in Chapter 5.

1.3 Motivation

1.3.1 Possible applications

Optimal scheduling of resources is a popular area of research with an almost inexhaustible list of possible application areas. Benefit can be derived by a number of industries from improvements in algorithms for the purpose. Scheduling was first applied in the industrial engineering realm for ensuring Just-in-Time delivery in factories by scheduling when unfinished materials were to be processed on each machine along the production line. Scheduling is very useful in Distributed Computing environments to allocate computing resources to users, or to schedule CPU threads optimally [6]. Scheduling has been used to find optimal task and time allocation strategies for the people working on large software projects, each with different skill levels for different tasks [7], and has also been used for scheduling of maintenance activities on electrical generators [8]. These are just a few examples.

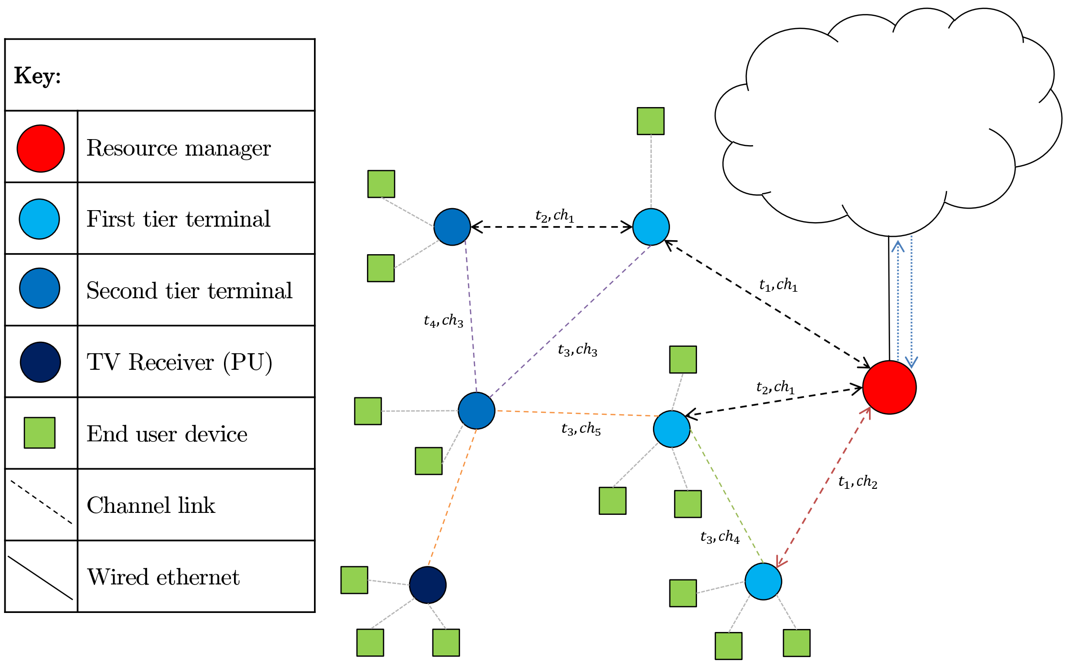


FIGURE 1.1: An example of a wireless network. The resource manager gathers transmission requests and allocates each requested flow a specific channel and time slot so as to optimise the overall capacity of the network

A very pertinent and topical application of this algorithm is in wireless communication networks where devices have limited access to channel and time slot resources. In recent years the issue of spectrum scarcity has become quite contentious and has sparked a fair amount of research activity into more optimal ways to allocate and use spectrum resources [9–11]. In order for varying Quality of Service (QoS) requirements to be met, different packets to be transported in the network may have different penalties related to their QoS level. Tardiness relates to throughput delay while, in such networks, if a packet is sent too early collisions may occur, thus prompting a series of retries and ultimately causing further throughput delays. Possibly, the destination may still be processing a previous frame and clearing its buffers, causing the frame to be lost and requiring further transmission retries. Limited buffer size may be a pertinent problem in low-power low-cost Wireless Sensor Networks. This may be the case in mesh and other networks where non-cooperative coexistence methods are employed. If, instead, the specific channel and time slot allocations to optimise these criteria are decided upon in advance by a channel or resource manager entity, collisions can be avoided and optimal use made of a channel as soon as it becomes available. Such a managed situation is illustrated in Figure 1.1. In the figure, t_i represent time slots and ch_i are wireless channels.

In most current wireless standards (such as IEEE 802.11af, 802.19.1, 802.16-2012, 1900.4), the MAC/PHY includes the architecture and data types to enable a managing entity such as the channel manager of 802.19.1 to allocate resources to devices based on certain objectives and using an on-board algorithm [12–14]. An application may be where nodes must first send RTS control frames containing the size of the data packet to the manager before continuing with the transmission. An example of a network with flows (links) allocated over time and channels (frequency) is illustrated in Figure 1.1. While protocol mechanisms are provided for the interaction mechanisms of the manager and its terminals to broadcast assigned channels, few specific algorithms are provided for the actual scheduling decision to be made. The algorithms presented in this work can be run by the management entity for network channel or time slot allocation, or both, in terms of IEEE 802.19.1, IEEE 802.22 [15] and other standards. The algorithm can be run on the channel manager node while waiting for channel availability and then when it becomes available, spectrum utilisation during the short assigned time slot can be improved. The algorithm may also be suited to wired networks.

1.3.2 Usefulness of more research on scheduling

While a fair body of literature exists in which SA is used to solve the single machine scheduling problem and a little has been done that deals with the specific optimisation criterion of weighted earliness and tardiness, most literature does not show in depth how the performance is influenced by the various parameters or quantify what performance can be expected for different combinations of values of the parameters, nor do they show how the parameters interact and influence one another. There is also no other work that applies and uses Metropolis-Hastings sampling in the unique way we do here or uses pre-sampling or hybridises sampling with SA to produce a more efficient algorithm. This fact emerges from the literature reviewed in Chapter 3.

The main contribution of this work is to add a pre-sampling step that enables a significant reduction in the search space before running the SA algorithm on scheduling problems in a way that enables low-cost solutions to be found much more efficiently than by the basic

SA algorithm alone. We present a unique application method and implementation of Metropolis-Hastings Monte Carlo to a problem that has not been done in the literature, as far as the author is aware, and that recognises and uses specific properties of the search space in solving this particular problem. We also investigate in detail the influence of parameters on the performance of the basic SA approach and introduce a new neighbour generation method that can be used in many meta-heuristic scheduling algorithms. This neighbour generation method relies on a new way of visualising the search space.

1.3.3 Why Simulated Annealing?

As is illustrated by the summary of previous work on the topic of heuristic algorithms for scheduling (Chapter 3), a number of algorithms have been used and have seen varying degrees of success. It has been claimed that the GA is a very, if not the most, successful meta-heuristic. GAs, however, have some disadvantages.

GAs are complex and require a number of further calculations and operations between iterations beyond the basic steps. For example, the mutation operator typically requires a number of operations and calculations to be performed to ensure the offspring created by the crossover operator are feasible [16–18]. These operations are typically complex. Additionally, the mutation operator must be applied either every iteration or when the best value found remains unchanged for a predetermined number of iterations. In the latter case, the operator must be applied as many times as the value remains unchanged, adding further complexity. In some cases a quantity such as the entropy within a generation is calculated first in order to determine the number of times the operator is to be executed [18]. All of these operations and calculations are particularly costly, adding time that is not affordable in today's highly time-sensitive applications.

In each generation a large number of solutions is considered and every time a new solution is created by the genetic operators, the chromosomes must be placed into their ranking order once again, necessitating a sorting algorithm to be run on the chromosomes of every generation. Even a relatively cheap sorting algorithm of $O(n \log n)$ run 1000 or even 100 times is very costly, particularly for large population sizes, e.g. where n is of the order of 200.

Adewole et al. performed a detailed comparison of SA and GAs as applied to the travelling salesman problem [19]. They conclude that GAs can provide a high quality of solution over SA but under the condition that the population set is large enough. Large population sizes in turn increase the complexity and running time exponentially. The researchers note that if powerful parallel computing capability is available, GAs can perform very well and reach solutions in shorter time periods. Noting our intended application area of wireless networks, this implies that GAs would be most useful in a fully distributed wireless network where all nodes have powerful computers on-board and can run parallel streams of populations. However, such fully distributed networks, where computing power can be shared, are not yet commonplace and are not the intended application scenarios for our work. Applications where GA meta-heuristics are chosen are typically less time-sensitive and more quality-centric. Examples are component placement in circuit design [20] or flight scheduling [21].

TS has also been used frequently but, as Raaymakers and Hoogeveen remark, requires significant tailoring to the specific problem [22].

In contrast, SA is generally applicable and relatively simple to implement. It requires only sampling a solution, calculating the Boltzmann quantity and comparing the result with a chosen random value. Solution quality provided by SA may not be as high as when using GAs but the running time is significantly lower. We have chosen SA as it is less complex and its implementation is simpler than GA but the method still offers the benefits of being able to escape from local optima and a minimal prior knowledge of the search space, while yielding acceptable solutions. In the intended application area of a centralised wireless network, obtaining a solution in a short space of time is more important than obtaining a solution that is very close to optimal (or is possibly optimal) but causes significant delays. The formulation of the problem as an SA problem is also well suited to the method we introduce to prune the search space.

1.4 Overview of Dissertation

The main contributions of this dissertation are to:

- Introduce a new neighbour generation method that can be used in meta-heuristic single machine scheduling algorithms.
- Introduce a new combination algorithm of Metropolis-Hastings pre-sampling and Simulated Annealing, which we call SAM, for the solution of single machine scheduling problems with due windows and without pre-emption, where idle time is not allowed, and set-up time is not required, and present a comparison of performance results from and theoretical computational complexity of basic Simulated Annealing, and the proposed hybrid algorithm.

In Chapter 2 we first go through some preliminary theoretical and foundational concepts, models and definitions to provide a sufficiently complete introduction to the work that follows. Chapter 3 presents some of the more pertinent literature related to this work. The design and details of my specific implementation of SA and a thorough description and development of the new hybrid algorithm of coarse Metropolis-Hastings sampling and SA is presented in Chapter 4. Chapter 5 contains a presentation and analysis of simulation results of the algorithms before we conclude in Chapter 6.

Chapter 2

Preliminaries: Models, Definitions and Techniques

2.1 Scheduling Theory

2.1.1 Notation and Definitions

The essential theory of scheduling that may aid understanding of the work presented in this dissertation is briefly presented here. This is drawn from some well-known literature in the field, in particular the definitive work by Pinedo [5], and is not intended to be any more comprehensive than necessary.

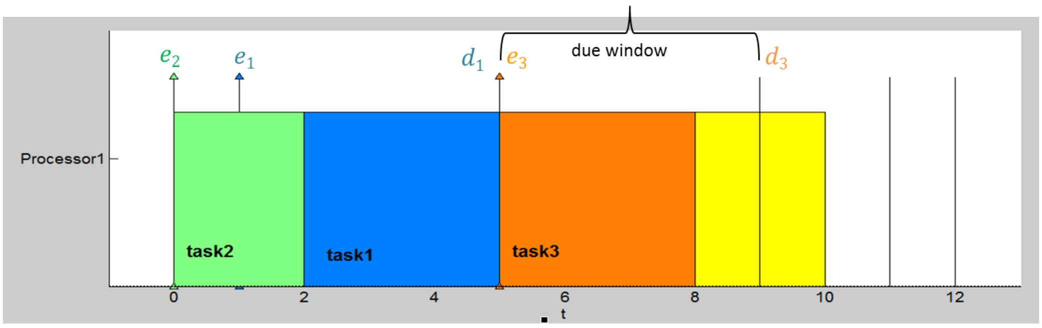


FIGURE 2.1: Example representation of a schedule, showing due window start and end times

Figure 2.1 illustrates elements of the general scheduling problem. The general scheduling problem is defined by a set of n jobs $J = (J_1, J_2, ..., J_n)$, which are to be scheduled on a set of m machines or processors $M = (M_1, M_2, ..., M_m)$. For the purpose of this work, it is assumed that a machine can only process one job at a time. If a machine is able to process a number of jobs simultaneously, this is called batch scheduling. Each job

$j \in J$ can be characterised by its processing time on machine i (p_{ij}), its due date (d_j) and a possible weight (w_j) or set of weights (w'_j and w''_j) for earliness and tardiness, which indicates relative importance. The job start time is denoted s_j. The subscript i denoting the processor is omitted in the single machine case. We generalise the due date into a due window within which a task must be completed to prevent carrying earliness or tardiness penalties. The beginning of the due window is given by e_j and the end of the due window by d_j as illustrated in Figure 2.1. The weighted earliness of job j is defined as $w'_j E_j$ and the weighted tardiness as $w''_j T_j$. The job's actual completion time is C_j. If pre-emption is not allowed, i.e. all jobs must be completed without interruption, we define $C_j = s_j + p_j$.

For a job with due window, earliness is defined as eqn. (2.1)

$$E_j = max\{e_j - C_j;\, 0\} \tag{2.1}$$

and tardiness as eqn. (2.2)

$$T_j = max\{C_j - d_j;\, 0\} = max\{L_j;\, 0\} \tag{2.2}$$

The schedule may have to allow for idle time on the machine(s) for maintenance, setting up or during faults or breakdown.

A schedule is feasible if no two jobs are overlapping and no job starts earlier than the schedule start time, i.e the condition of eqn. (2.3) is met.

$$\{s_i \geq C_j \ \forall \ i > j\ \} \wedge \{s_i \geq t_0\} \ \forall \ i, j \in J \tag{2.3}$$

The scheduling problem is described using the traditional three-field notation, $\langle\ \alpha \mid \beta \mid \gamma\ \rangle$ [1], where

- α describes the processing environment. For single machine problems, this value is simply 1.
- β describes the constraints. These could include a release date r_j when the job initially becomes available for processing, set-up time, precedence constraints, pre-emption or any of a variety of other constraints.
- γ is the objective function. Some examples are the makespan (maximum completion time), total completion time (the sum of completion times of all jobs) or total weighted completion time. The total earliness and tardiness of jobs, either

separately or jointly (defined in eqn. (2.4)), are also common functions to be minimised (and the objective function of this work).

$$\gamma = \sum_{j \in J} (w'_j E_j + w''_j T_j) \tag{2.4}$$

Numerous models of the scheduling problem exist, each having very different methods that may provide the best results. Some variants to the model include situations where set-up time is considered as a cost separately from the processing time, where pre-emption may or may not be allowed, and where processing is either batched or non-batched. Schedule models with several processors include parallel-machine, where machines may be identical or differing in capabilities and/or speeds; flow-shop, where each job has to go through a series of different processors before being complete; and flexible flow-shop, a generalisation of flow-shop; job-shop, which is still further abstracted in that each job may have an independent set of processes to follow, and open-shop processing, where jobs are to be processed on each machine in any order. Models may define deterministic job characteristics or may be stochastic, such as in the Markov Decision Process approach [23]. A newer model that has emerged is multi-agent scheduling, where each agent is responsible for a certain set of jobs and has its individual set of objectives to optimise.

With so many more complex models in existence, it may be necessary to justify the practical applicability of the model we have chosen for our analysis. A large number of practical problems are in fact accurately reflected in the single machine model [5, 24–26]. Examples may include a number of tasks or task segments to be processed by a single CPU. Even if there is more than one CPU in such a situation, the units often function independently in parallel and thus are decomposed into separate single processor problems and solved independently. The decomposition of multi-machine problems into single machine problems is often done [5] and so this model provides a good fundamental basis for analysis, and results can easily be extended to apply to more complex problems. The situation where pre-emption is not allowed has been chosen as this is also a generalised form of the problem, and the model can easily be adapted to situations with pre-emption allowed. For example, jobs can be divided into arbitrarily small pieces and scheduled as separate tasks to model situations where pre-emption is allowed. The criteria of weighted earliness and weighted tardiness are very common measures in industrial settings and several other just-in-time applications, and are recognised in the literature as important for research focus [4, 16, 27–31]. The problem as it stands is already known to be strongly NP-hard [4, 26, 32–34]. In fact, even the problem considering only one objective $\langle 1 || \sum_{j \in J} w''_j T_j \rangle$ is known to be a strongly NP-hard problem [5]. Thus, our problem is complex and practically applicable enough to justify investigation, and to justify investigation into evaluating and improving the performance of meta-heuristic algorithms to solve this type of problem.

2.1.2 Priority dispatch rules

Some theorems on the prioritisation of tasks, known in the literature, are briefly presented here. These may be optimal for certain simpler single-criterion scheduling problems but are not optimal for the problem of this work. We present them as they form a starting point for a number of solutions presented in the literature.

Theorem 2.1 (Shortest Processing Time (SPT) [5])**.** *When minimising completion time, an optimal schedule is found by sorting jobs in increasing order of processing time.*

This is called the *Shortest Processing Time (SPT)* and *Shortest Remaining Processing Time (SRPT)* rule, which also extends to *Weighted Shortest Processing Time (WSPT).*

Proof. The reason emerges from the equation for total completion time shown below for the unweighted case (eqn. (2.5)) and weighted case (eqn. (2.6)). These equations show that when scheduling N jobs, the first job's processing time is added N times, p_2 is added $N-1$ times, p_3 added $N-2$ times and so on, so to minimise the total completion time it is necessary to start with the shortest, and end with the longest processing time.

$$\sum_j C_j = (t+p_1) + (t+p_1+p_2) + (t+p_1+p_2+p_3) + ... \tag{2.5}$$

$$\sum_j C_j = (t+p_1)w_1 + (t+p_1+p_2)w_2 + (t+p_1+p_2+p_3)w_3 + \dots \tag{2.6}$$

□

Corollary 2.2. *The WSPT heuristic processes the values $\frac{p_j}{w_j}$ in non-decreasing order, where w_j is the weighting given to job j, as shown by eqn. (2.7):*

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq ... \leq \frac{p_n}{w_n} \quad . \tag{2.7}$$

Building on from these rules, certain problems may enable the assumption of restrictions on processing times and weights. These theorems are presented and proved by Yano and Kim [27]. These are:

Theorem 2.3. *For adjacent jobs i and j with starting time $s_i \geq s_j$, it is optimal to sequence i before j if and only if the conditions given by eqn.* (2.8) *to* (2.11) *are met.*

$$E_i/E_j \leq p_i/p_j \tag{2.8}$$

and

$$p_i/p_j \leq T_i/T_j \tag{2.9}$$

and

$$E_ip_i + T_jp_i \leq (T_j + E_j)(s_j - s_i + p_j) \tag{2.10}$$

or

$$E_i + T_i \leq T_j + E_j \tag{2.11}$$

Proof. The proof is presented in [27]. □

Theorem 2.4 (Earliest Due Date (EDD) [35])**.** *Jobs are to be sorted into increasing order of due dates to obtain an optimal schedule for minimising maximum lateness or maximum tardiness.*

Proof. Full proof in [35]. Intuitively, jobs with earlier due dates are in greater danger of being later than their due date than jobs with later due dates, so to obtain an optimal schedule, these must be scheduled first. Where jobs are weighted, the weighted due date must be used. □

Theorem 2.5 (Least Cost Last (LCL) [36])**.** *Jobs with the lowest weighted cost are scheduled towards the end to solve the general $\langle 1|prec|f_{max}\rangle$ in $O(n^2)$ time.*

Proof. Full proof in [36]. As the rule is an extension of EDD, similar to the EDD rule, this aims to minimise the total cost of the schedule. □

The techniques above, and extensions to these, are heuristic techniques rather than algorithms and are specific to scheduling problems. They are called "constructive" since they gradually build up a schedule using certain rules [5].

2.2 Algorithms

Meta-heuristic algorithms find either approximate solutions to optimisation problems, or solutions to not all instances [37]. Most do this by starting with an initial solution and then searching the space of all possible solutions to improve on it. Meta-heuristic algorithms such as Genetic Algorithms, Simulated Annealing and Tabu Search have been used to solve scheduling problems but can be applied to any of a wide variety of other optimisation problems. Deterministic or exact algorithms in contrast can be used to find the optimal solution but their time complexity for large problems may be unacceptably high in cases. Two of these optimal methods are dynamic programming and the branch-and-bound algorithm. Branch-and-Bound is often cut short to reduce computation resulting in only a suboptimal solution, when the gap between the upper and lower bounds becomes acceptably small. Complexity is a key consideration in the choice of algorithm. We formalise the notion of complexity classes and appropriate definitions in Appendix A.

2.3 A Note on Notation

Some points about the notation used are necessary to avoid ambiguity. I have tried to be as consistent as possible in this respect. Random Variables (R.V.s) are given as capital letters (e.g. Y and X) unless otherwise stated. Sample spaces are given by similar notation. The terms observation, random draw and sample are equivalent.

i) A single observation of a scalar R.V. is denoted e.g. x, y or, explicitly, $X = x$.

ii) A set of n random draws from a scalar R.V. is denoted as $x = \{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$.

iii) A row vector of size D is written $\mathbf{y} = (y_1, y_2, ..., y_D)$.

iv) The nth sample of the vector R.V. $\mathbf{Y} = \mathbf{y}$, is $\mathbf{y}^{(n)} = (y_1, y_2, ..., y_D)^{(n)}$.

v) A set of n random draws from a multidimensional or vector R.V. is in bold print e.g. $\mathbf{x} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ...\mathbf{x}^{(n)}\}$.

vi) The probability density function (p.d.f.) of a random variable over the set of all possible values is written in uppercase $P(X)$ or P_X, while the probability that the random variable takes on a specific value is written in lowercase $p(X = x)$.

vii) A set of values that occupy d-dimensional space may be denoted $S^d \subset \mathbb{R}^d$.

viii) The term "distribution" is used in this context to mean p.d.f. P and the terms are used interchangeably. This is in keeping with the usual literature on the topic, though it is noted that "distribution" may also refer to the cumulative distribution function in other texts.

ix) Conditional expectations will use the notation:

$$\mathbb{E}_{\pi(x|y)}[f(x)]$$

for example, which is equivalent to

$$\mathbb{E}_x[f(x)|y] = \int_x p(x|y)f(x)dx \quad .$$

x) One final note applicable in particular to the results (Chapter 5), where we make a perhaps sacrilegious return to frequentist techniques for evaluation and results analysis, is that a p-value of 0.05 or less for statistical significance tests such as Chi-squared and t-test, is considered statistically significant.

2.4 Bayesian Inference

Bayesian probability is an integral part of all Machine Learning algorithms, since it provides a method by which inference about the future can be made from previous observed data or about latent variables, given observed data variables [38]. Bayesian probability exploits dependencies between data samples, rather than assuming or forcing an artificial independence. The application of Bayesian statistics to MCMC has been directly responsible for the great popularity that MCMC has seen in recent times, enabling characterisation of complicated posterior distributions such as parameters in complicated system models [39].

Scalar R.V.s are used to illustrate concepts for clarity and to avoid the messy notation of Jacobians that would be required otherwise. Bayes' theorem states that, given sample data $y \in S_1 \subseteq \mathbb{R}$ from a distribution with known likelihood $P(y|x)$, a *posterior* distribution $\pi(x|y)$ (given by eqn. (2.12)) for unknown variable $x \in S_2 \subseteq \mathbb{R}$ can be inferred from an *a priori* density for x given by $P(x)$ via the conditional probability, as shown in eqns. (2.12) and (2.13):

$$\pi(x|y) \propto P(y|x)P(x) \tag{2.12}$$

where the constant of proportionality is

$$\int_{x \in S_2} p(y|x)p(x)dx = p(y) \quad . \tag{2.13}$$

The expression $P(x)$ captures initial assumptions about x before the observation. The equivalent for discrete random variables, given by eqn. (2.14), is equally valid.

$$\pi(x|y) = \frac{(P(y|x)P(x))}{P(y)} = \frac{P(y|x)P(x)}{\sum_{S_2}(p(y|x)p(x))} = \frac{P(y|x)P(x)}{\sum_{S_2}(p(y,x))} \tag{2.14}$$

where $p(y,x)$ represents the joint probability of x and y. We may want to find expectations, such as eqn. (2.15), of a function $f(x)$ over the probability density $\pi(x|y)$:

$$\mathbb{E}_{\pi(x|y)}[f(x)] = \int_{S_2} f(x)\pi(x|y)dx \quad , \tag{2.15}$$

or the discrete equivalent. Clearly, we require that $f(x)$ is integrable with respect to $\pi(x|y)$. For many problems it may not be feasible to evaluate the integral directly, either because no closed form integral exists or owing to high dimensionality making computation practically prohibitive. In such situations, it is necessary to employ approximations or relaxation techniques. Such problems are very common and this is where Monte Carlo techniques shine.

2.5 Markov Chains

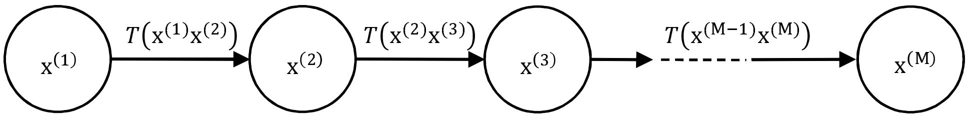


Figure 2.2: A Markov Chain representing states of a Monte Carlo random variable with transition probabilities

A Markov Chain, such as illustrated in Figure 2.2, is a series of states or random variables with memory representing a stochastic process so that the value or probability of a state depends on previous state(s) and the dependency relationships can be represented by transition probabilities, T_m. If a state depends only on its immediate predecessor, the Markov Chain is said to be first order; if it depends on its immediate predecessor as well as the one before, the chain is second order and so forth. More formally, for a first order Markov Chain $x^{(1)}, x^{(2)}, ..., x^{(M)}$, the condition of eqn. (2.16) holds.

$$\begin{aligned} P(x^{(m+1)}|x^{(1)}, ..., x^{(m)}) &= P(x^{(m+1)}|x^{(m)}) \\ &\equiv T_m(x^{(m)}, x^{(m+1)}) \\ \forall\, m \in \{1, ..., M-1\} &\subseteq \mathbb{N} \end{aligned} \tag{2.16}$$

For the purpose of MCMC simulation, we require that the chain is *ergodic*, that is, it is required that the distribution $P(x^{(m)})$ converges (asymptotically) to a single invariant distribution (the equilibrium distribution) when $m \to \infty$, regardless of the chain's start point. Provided that the conditions on the transition probability are met, the invariant or stationary distribution satisfies eqn. (2.17).

$$P(x) = \sum_{x'} T(x', x) p(x') \tag{2.17}$$

2.6 Monte Carlo Simulation

2.6.1 Ordinary Monte Carlo

A number of good texts exist on this topic, which the reader may consult if necessary. We have drawn from Christopher Bishop's comprehensive text here [38], as well as [39–42], and others cited as needed.

Ordinary Monte Carlo aims to estimate properties, such as expectations, of a certain p.d.f. $p(x)$ by drawing N independent identically distributed (i.i.d.) samples from the distribution of random variables and approximating the density function using a point mass approximation for the density [43], as shown in eqn. (2.18).

$$P(x) \approx P_N(x) = \frac{1}{N} \sum_{i=1}^{N} \delta_{x^{(i)}}(x) \tag{2.18}$$

Then, expectations are calculated using the usual averaging calculation,

$$\mathbb{E}_N[f] = \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}) \quad , \tag{2.19}$$

which tends to the required expectation

$$\mathbb{E}[f] = \int f(x) p(x) dx \tag{2.20}$$

in the limit $\lim\limits_{N\to\infty} \mathbb{E}_N[f]$.

Now if

$$\mathbb{E}_{P(x)}[f^2(x)] - (\mathbb{E}[f])^2 < \infty$$

holds, and noting that for ordinary Monte Carlo samples are i.i.d., we can use Central Limit Theorem to find a normal p.d.f. of the approximation's error

$$\mathbb{E}_{P_N}[f(x)] - \mathbb{E}_P[f(x)] \quad . \tag{2.21}$$

The problem, however, is that all too often it is not feasible to get i.i.d. samples from the required distribution and expectations are intractable or impossible to calculate. That is where Markov Chain Monte Carlo comes to the rescue.

2.6.2 Markov Chain Monte Carlo

The essence of MCMC is that statistical inferences can be made about a system without having to know, or be able to simulate, the exact behaviour of a system or know the form of a function describing the behaviour, and this can be achieved with dependent samples instead of i.i.d. samples. We only need to construct a Markov Chain that has the same equilibrium distribution as the true system in question and employ some Bayesian statistics. The complicated distribution and the constructed distribution are typically related by a constant that is difficult to calculate or for which there is insufficient data to calculate, such as may arise in finding a normalised Bayesian posterior. MCMC nonetheless enables one to gather statistical information about the true system, such as expectations of functions under the distribution. This powerful technique performs well even in the case of high dimensionality.

Suppose we wish to sample from a complicated, unknown or unknowable unnormalised distribution $\pi(x)$. We are able to evaluate $\pi(x)$ for any given x up to a certain constant, X_p. This constant would normally involve a complicated integral from the Bayesian inference step and is not known. So we have

$$\pi(x) = \frac{P(x)}{X_p} \quad .$$

We cannot sample from $\pi(x)$ directly so we construct a Markov chain on $x \in S \subset \mathbb{R}$ with incremental transition probabilities $T(x, dy)$, where both x and $y \in S$. Marginalising x, the resultant distribution becomes

$$\int_{x\in S} p(dx)T(x,dy)dx = \int_{x\in S} p(dx)q(dy|x)dx = P(dy) \quad , \tag{2.22}$$

which is the Bayesian normalising constant. We can now sample from a simpler proposal distribution, the transition probability $Q(x^{(i+1)}|x^{(i)})$. Each time a sample is drawn from the proposal distribution it is recorded for comparison with the next sample. The distribution as well as the next state drawn thus depends on the current state. This is the Markov Chain. As the number of samples becomes very large, $P(dy)$ approaches $\pi(x)$. The choice of proposal distribution is arbitrary but its choice influences how fast the chain converges to the desired stationary distribution while the specific update mechanism determines whether a candidate state is accepted or not. In choosing Q there are several subtleties. Details about the conditions to ensure the chain converges to the equilibrium distribution are discussed in Section 2.6.2.3. It is sufficient to say for now that it is usually possible to construct a Markov Chain with the required properties.

Several sampling methods are harboured under the umbrella of MCMC. These include rejection sampling, importance sampling, slice sampling and Gibbs sampling. The focus of our work is on the Metropolis-Hastings method discussed below. One of the reasons for this choice is that it allows for an asymmetrical proposal distribution. Another widely used method, Gibbs sampling, is generally applied to a different problem type where the variables represent parameters of a statistical model to be estimated. The parameters must be able to be partitioned into independent sets and should follow a standard conditional probability density function. In this method all proposed updates are accepted. Not only is our situation not one of parameter estimation and so not well-suited to Gibbs sampling, but the acceptance of all updates increases the required memory to store all states, and the running time. Gibbs sampling is simply a special case of the Metropolis-Hastings algorithm, as is rejection sampling [44]. We prefer the more general method.

Importance sampling is another option, but this is a method only for estimating the expectation of a function and does not generate samples from the distribution [45]. We would like samples to be generated for use in the SA step. Another disadvantage of importance sampling is that there may be a very large error in the estimate since the empirically found variance does not necessarily represent the actual variance to any degree of accuracy. Slice sampling may have been an advantageous option, but Metropolis-Hastings is clearer and simpler to implement.

2.6.2.1 Metropolis-Hastings

The Metropolis-Hastings method provides a way of updating samples in the random walk of a Markov Chain Monte Carlo simulation. We start by specifying three distributions, using the variable z here:

i) Target distribution is the approximation to the desired distribution which we may know up to the normalising constant. This is $\tilde{P}(z)$.

ii) Proposal distribution. This is an arbitrarily chosen starting transition probability $q(z'|z)$ used to draw new samples. The aim is to construct the chain so that the samples drawn look like samples from the actual distribution, $\tilde{P}(z)$.

iii) A function that samples from the proposal density.

Now, supposing the first state is $z^{(\tau)}$, a candidate point $z^{(*)}$ is generated from the proposal distribution. The candidate is accepted as the new state with probability given

$$\alpha(z^{(*)}, z^{(\tau)}) = min\left(1, \frac{p(z^{(\tau)})q(z^{(\tau)}|z^{(*)})}{p(z^{(*)})q(z^{(*)}|z^{(\tau)})}\right) \quad . \tag{2.23}$$

In practice, the choice is achieved by comparing the value of $\alpha(z^{(*)}, z^{(\tau)})$ with a value drawn from the uniform distribution in the interval $(0, 1)$ and accepting if $\alpha(z^{(*)}, z^{(\tau)})$ is greater than this sample. If the candidate is not accepted, $z^{(\tau+1)} = z^{(\tau)}$. We then draw another sample from the proposal distribution and repeat until the specified terminating conditions are reached. Theoretically, as $\tau \to \infty$, $P(z^{(\tau)})$ converges to the desired distribution. Convergence is discussed further in Section 2.6.2.3.

First, a note on the choice of proposal distribution: The success or failure of the MCMC simulation hinges quite heavily on the choice of proposal distribution. If it is too narrow, a large proportion of candidate samples will be rejected and a large number of iterations may be required before any useful information is gained from the sample. In contrast, if it is too wide, a large proportion of samples are accepted and correlations may be high, once again limiting the information conveyed by the samples. The choice of proposal distribution may be the most important decision for an implementer to make, and is the one for which no guidance truly exists.

The existence of a stationary distribution is a sufficient condition for irreducibility. This condition informs the choice of proposal distribution, since the proposal must have a wide enough girth to support the probability of reaching any state in the state space with a positive probability in π. Both the blessing and the curse is that this guidance on choosing a proposal distribution is not much guidance at all since almost any arbitrary distribution chosen will fulfil the requirement. On the other hand, most distributions may take an insupportably long time before the chain starts to converge.

MCMC is a supremely powerful technique allowing the characterisation of complex systems but one runs the risk of wasting resources with a too-lengthy run or never reaching the equilibrium state. We wish to exploit the power of the system while limiting exposure to, or consequence of, this weakness. Our question is whether it is useful to use the technique to pre-sample and get a general idea of the terrain of the problem's search space to inform a more focused search, and if this can be done in a way that saves rather than compounds overall complexity when in combination with a local search heuristic. Such questions are addressed in Chapters 4 and 5, but are greatly influenced by the implementation issues we now put forward.

2.6.2.2 Burn-in

A common practice in MCMC is to include a "burn-in" or "thermalisation", a set of samples drawn and discarded before samples are recorded for use in obtaining the required distribution. The reason for this is usually asserted to be removing biases caused by high initial correlation between samples. Geyer asserts, however, that what is important is rather to find a good starting point for the chain [41]. He asserts that if we could start the chain somewhere in the middle of the equilibrium distribution, there would be no bias to eliminate, and since there are other methods that can be used to find starting points, there is no theoretically defensible reason for the practice [41]. Unfortunately, if there truly is no knowledge of the posterior distribution there is no way to ensure the starting point fulfils this requirement (or hope). Yet burn-in is still an accepted, most common and expected addition to the Metropolis-Hastings initialisation process.

2.6.2.3 Convergence

To ensure that the chain does in fact converge, it is required that the system has the property that starting from any point (vertex if we consider the chain a graph) in the state space, an infinite random walk will always end at a certain point v. Since the Markov Chain constructed is ergodic and stationary with respect to $\pi(x)$, if $X^{(t)} \sim \pi(x)$, necessarily $X^{(t+1)} \sim \pi(x)$ for all t, so X is converging to π. The condition of general balance (eqn. (2.24)) must hold for a stationary distribution π, i.e.

$$\pi T = \pi \quad , \tag{2.24}$$

where T is the transition probability matrix, which leaves the distribution unchanged.

Restrictions on the transition probability to ensure convergence (and ergodicity) are:

- **Connectedness**, which implies **irreducibility** - there is a positive probability of transitioning from any state to any other state. The method requires connectedness so that the same result can be obtained regardless of where the chain starts.
- **Aperiodicity** - intuitively a periodic function implies that the chain does not end while we require a finite chain.

When using Metropolis-Hastings method, one constructs a number of transition probability matrices along the chain, each individually maintaining π but not usually individually irreducible with respect to the state space S [39]. In the case of a chain of such transition probability matrices, we require eqn. (2.25):

$$\sum_{x \in S} \pi(x) T(x, x') = \pi(x') \tag{2.25}$$

to hold for $x, x' \in S$. This condition is both necessary and sufficient to prove invariance. However, since the sum may be intractable to solve, the more stringent condition of detailed balance may be introduced as a sufficient condition. If detailed balance holds,

$$\pi(x) T(x, x') = \pi(x') T(x' x) \quad . \tag{2.26}$$

If each $T_1, ..., T_n$ individually maintains detailed balance, so does T. Detailed balance also implies that the chain is time reversible and that Metropolis-Hastings then reduces to a Metropolis sampling.

The difficulty with MCMC methods is that the resultant distribution can only be claimed conclusively to have converged to the equilibrium distribution after infinity iterations. This means there are no better than approximate ways to calculate when to stop the chain of the simulation, and there is no way to know how close an approximation the result is to the real desired distribution.

A number of heuristic techniques are thus required to inform the decision of where to stop the chain. There are a number of different methods which can be compared in order to determine a suitable point of termination. Some are presented below.

Gelman and Rubin

Gelman and Rubin's method requires that a number of independent parallel chains are run, all with different starting points [46, 47]. A factor is then calculated that quantifies how a parameter might shrink if sampling were to be continued indefinitely. The factor includes a comparison between variance of the means of the m parallel chains (B), and the m averages of the variances within the m chains (W). If the scale factor is close to one, they posit that this means the chains are effectively unbiased by the starting points and likely to approximate the target density. The method involves two steps:

i) Before sampling, it is required that an "over-dispersed" estimate of the target density is obtained so that suitably spread out starting points for parallel chains may be generated.

ii) For each quantity, the last n iterations are used to estimate the target distribution of the quantity using a Student-t distribution.

Using the symbols above, for m parallel chains, where df represents the degrees of freedom, the factor is calculated as

$$\sqrt{R} = \sqrt{\left(\frac{n-1}{n} + \frac{m+1}{mn}\frac{B}{W}\right)\frac{df}{df-2}} \quad . \tag{2.27}$$

While Gelman and Rubin's method has seen reasonable popularity, there are a few points worth questioning. Firstly, it is computationally very inefficient to run a number of independent chains when the same computation time may have been used simply to run a single chain for longer and possibly get a closer approximation to the target distribution. Secondly, the first step of getting over-dispersed samples from the target implies that there must be knowledge of the limits of the sample space, which is unlikely if the point of the MCMC simulation is to infer properties of a distribution without any direct knowledge about it. Thirdly, it involves running the simulation for a certain number of iterations, and then using the last n to calculate the diagnostic factor. The choice of n encounters the same problem with which the original Monte Carlo method is afflicted. It may be necessary to repeat this step several times and compare how the factors shrink, costing extra time and computation, which may have been used with equal gain to run the Markov Chain for longer.

Raftery and Lewis

The Raftery and Lewis method assists in determining the length that the Markov Chain in MCMC algorithms should reach, determined by the properties of a short initial "pilot" run [46, 47]. The premise is that the ergodic average of the chain is asymptotically normal for large number of samples [44] by the central limit theorem. They estimate the value of n that ensures that $P(\theta - \epsilon \leq \overline{\theta}_n \leq \theta + \epsilon) = 1 - \alpha$ for deviation $\epsilon > 0$ and $0 < \alpha < 1$ by using the formula,

$$n \approx \widehat{var}(\overline{\theta}) \left[\frac{\Phi^{-1}(1-\frac{\alpha}{2})}{\epsilon}\right]^2 \quad , \tag{2.28}$$

where $\Phi(\cdot)$ here denotes the normal cumulative distribution function and $\widehat{var}(\overline{\theta})$ is an estimate of the variance of the ergodic average, based on the observations gained from the pilot run.

Geweke

Used to determine when to terminate an MCMC sampler when MCMC is used to find the mean of a function $g(\theta)$, this method assumes that the function g has a spectral density that is continuous at a frequency 0 [46]. A set of MCMC samples after N iterations can be viewed as a time series. Owing to the properties of Markov processes, the Geweke method considers this a Wide Sense Stationary signal and uses the relationship between the variance of a time series and its power spectral density $S_g(\omega)$ (where ω is angular frequency). After N iterations, the estimated expectation of a function g with respect to θ, is given by

$$\bar{g}_N = \frac{1}{N}\sum_{i=1}^{N} g(\theta^{(i)}) \quad . \tag{2.29}$$

The asymptotic variance of the series is then given by $S_g(0)/N$, the power spectral density over N. This estimated standard deviation calculated from the variance is called the Numeric Standard Error (NSE). This value can be calculated at several points until the desired precision is achieved.

The convergence diagnostic functions by taking the difference between means calculated on N_A iterations from the beginning of the chain, $\bar{g}(\theta)_N^A$, and $\bar{g}(\theta)_N^B, N_B$ iterations from the end, and computing

$$\frac{\bar{g}(\theta)_N^A - \bar{g}(\theta)_N^B}{NSE} \tag{2.30}$$

Geweke suggests N_A to be 1/10 of the total iteration count and N_B to be half. Various implementations of the Geweke diagnostic exist, one being to calculate correlations between the two subsets of the total number of samples and determining whether they are sufficiently different.

2.7 Simulated Annealing: The technique

SA is a probabilistic search heuristic used in optimisation problems with complex search space characteristics, such as multiple local optima, first introduced by Kirkpatrick, Gelett and Vecchi [48]. The algorithm models a search on the physical process of annealing where a substance is slowly cooled so that freezing occurs at the minimum energy configuration (the, or a, minimum). General requirements are a finite search space, a real-valued cost function, a set of neighbours for each state or neighbour generation method and a non-increasing temperature cooling function. Starting from the first state, the algorithm creates a discrete Markov Chain with the evolution of states in the chain determined on the basis of

i) whether the cost has decreased

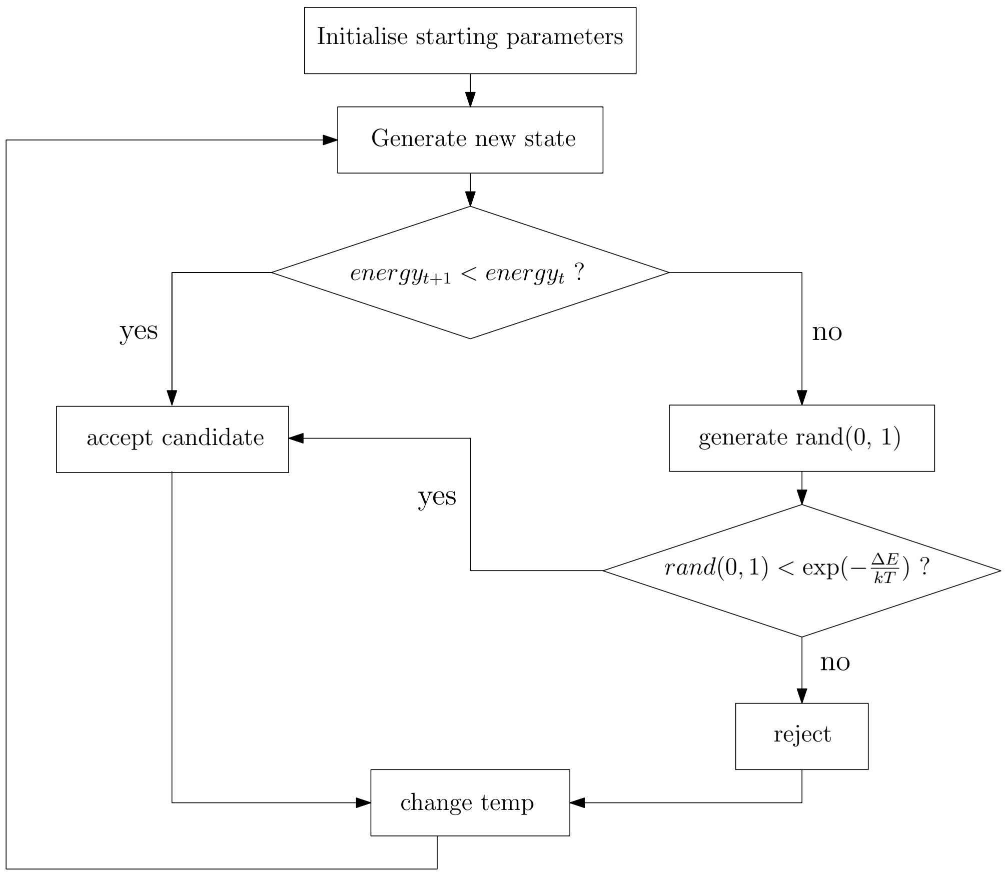

Initialise starting parameters
Generate new state
$energy_{t+1} < energy_t$?
yes
no
accept candidate
generate rand(0, 1)
yes
$rand(0,1) < \exp(-\frac{\Delta E}{kT})$?
no
reject
change temp

FIGURE 2.3: Flow chart showing general steps of the Simulated Annealing algorithm.

ii) if the cost of the new state is not lower than the previous value, the state value may still be accepted using a Boltzmann probability distribution. This feature enables the algorithm to escape from local optima.

The general SA algorithm is given in Figure 2.3. The block rand(0, 1) represents the generation of a random real number from a uniform distribution in the range 0 to 1 (excluded). While the SA algorithm is generic, some important aspects of the implementation are mentioned here. Detail about the generation of pseudo-random numbers is important as unintentional biases that may be introduced within the generation method may have unfavourable consequences. The perturbation function for generating neighbour states, to be determined by the implementer, depends on the application and is instrumental in the performance of the algorithm.

That concludes all the main background theory and applicable technical information that forms the basis for all the work that follows.

Chapter 3

Related Work

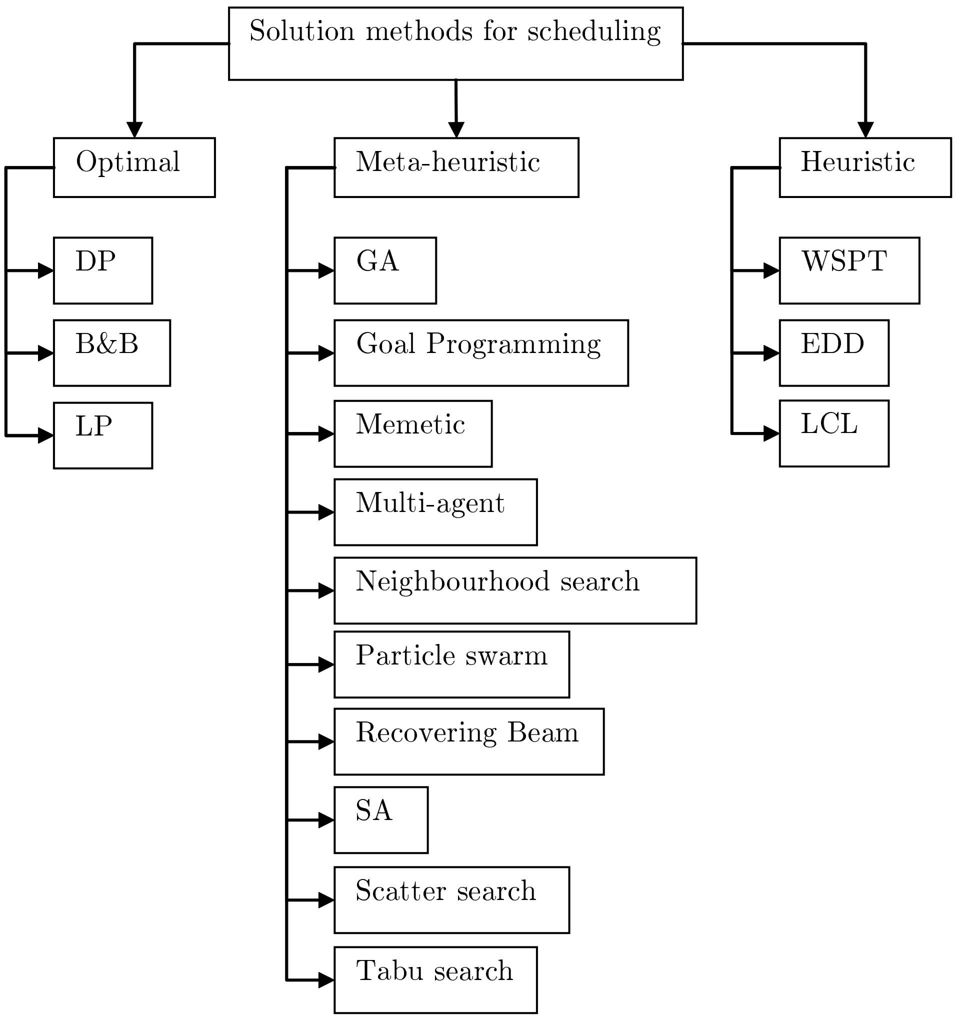


FIGURE 3.1: Taxonomy of solution procedures for scheduling problems

This section is structured in a top-down manner, first providing an overview of the main classes of algorithm used in the available literature to solve scheduling problems as well as a clear taxonomy, and then detailing the approaches in subsequent sections. In addition to the better known algorithms applied to the scheduling problem, some different and specific methods have been developed. These are discussed briefly. The final section investigates comparative works where the performance results of different algorithms have been compared. While the focus is on single machine scheduling, other works that have investigated multiprocessor scheduling are also mentioned.

Generally, solution methods may be divided into optimal or exact, and suboptimal or heuristic methods. Heuristic methods include problem-specific and more general meta-heuristic techniques for producing suboptimal results. The taxonomy of algorithms that are discussed in this dissertation is given in Figure 3.1.

3.1 Simulated Annealing

Simulated Annealing has usually been used in the scheduling literature in cases where there are other complicating factors to the problem. The objectives for the algorithm vary, as do the applications and number of machines. It is rare to find literature on SA being used for the basic single machine scheduling problem with weighted earliness and tardiness penalties. Some only consider a single objective, or investigate multiple machine scheduling, learning effects or more application-specific objectives.

Tan and Narasimhan investigate minimising tardiness on a single machine with the addition of sequence-dependent set-up times [49]. A rather unique situation is investigated by Jozefowska et al., where each task can be executed in one of several modes, which are divided according to activity resource requirements and duration [50]. They use SA for minimising the makespan with pre-emption not allowed. Their process involves assigning modes to jobs and feasible starting times for activities, resulting in an activity list (schedule) and a list of execution modes. The researchers perform pre-processing on the search space to eliminate non-executable or inefficient modes and redundant non-renewable resources. Multiple machine single objective scheduling is approached using SA by Kim et al., where they also consider the additional aspect of set-up times between jobs of different lots [51].

3.1.1 Multi-objective

Similar to our work, Mahdavi et al. address the single machine scheduling problem to minimise total weighted earliness and tardiness without pre-emption also with an SA approach [52]. In their formulation, however, they assume controllable processing times where job lengths can be expanded or compressed within certain limits. They seek to find an optimal set of expansions, compressions and job sequence. They also assume fixed due dates and not due windows as we do. Priority dispatch rules are employed to find a suitable starting point for their SA algorithm using the net-benefit-compression — net-benefit-expansion heuristic. An application-specific multi-objective problem is approached by Saraiva et al. for generator maintenance [8].

An attempt is made to find a general solution to multi-objective scheduling problems with two or more objectives by Loukil et al. [53]. Their method requires that a set of "potentially efficient" points be initialised at the start and a family of weighting functions be defined to direct the statistical search to potentially more efficient solutions. These functions are used every iteration of the algorithm to update the set of potentially efficient points. This adds complexity to the method and it can be seen that the choice of weighting functions is both arbitrary and problem-dependent and can have a large impact on the solution. The authors also concede that a large number of experiments is necessary to determine the number of sets of weights that give "good" solutions. The method also involves a filtering procedure where the solutions in potentially efficient sets undergo pairwise comparison to remove solutions that area unfairly dominated by any one of the objectives. This requires n choose 2 operations. It is our opinion that this solution is unnecessarily complex and contains too many variables that have a large impact on the solution quality and run time, and that there is space for more innovation in multi-objective scheduling algorithms.

3.1.2 Learning effects

Wu et al. use SA for the single machine scheduling problem where the objective is to minimise total lateness of work, which is somewhat related to our objective [54]. Their contribution is to include a job position-based learning effect. This work is continued by Wu in solving a two-agent single machine scheduling problem, in which two or more distinct families of jobs with different criteria are the subjects for scheduling [55]. Learning effects in the context of transmission mode selection in LTE Networks are considered by Wang et al. [56]. Fichera et al. also consider learning effects, as well as deterioration effects, in factories; specifically, the effect of workers having to learn and adapt to different lots of products, and machine deterioration [57].

3.1.3 Neighbour generation

Neighbour generation is known to be a strategic part of the implementation of SA, having a profound influence on the performance [8], yet most frequently in the literature the same methods are employed i.e. selecting random positions and performing various shift, swap, and insertion operations on tasks. Fichera [57], Kim et al. [51], Loukil et al. [53], Jozefowska et al. [50], Tan and Narasimhan [49] and Wu et al. [54] are examples that all use interchange and insertion of items. These methods both involve unnecessary complexity and add to the algorithm run time, necessitating the storage and generation of a number of arrays and performing numerous operations on them many thousands of times, failing to take advantage of the general structure and pattern of the costs of schedules when arranged in a lexicographical order.

3.1.4 Experimental methodologies and results

The most common problem instance generation technique used in the literature is that proposed by Potts and Van Wassenhove [58] and expanded by Hall and Posner [3]. This generation technique is considered the classical approach [54, 59] and is also the method we use in Section 5.1.1. Other researchers use real industry problems [8, 22] or benchmark problems from libraries [50]. Unfortunately, the libraries are not open access or are no longer available so we could not use these. They also appear not to have variable due windows.

Mahdavi et al. generate problem sets with due dates from a uniform distribution [52]. In their experiments they also measure performance by relative percentage deviation from the optimum. SA is run only five times on each problem instance to account for statistical variation. As is shown in the results section of our work (Chapter 5), statistical variation may be large enough that five runs is not adequate to find acceptably representative averages.

Initial temperature is an important parameter to fix and is often determined by setting a starting acceptance ratio and deriving temperature from that [51]. It has been asserted that initial solution is also an important factor in algorithm performance and much of the literature sees the use of priority dispatch rules or other methods to generate a good starting solution [22, 51, 60]. The effect of starting point on the final solution is investigated by Tan and Narasimhan, using “better than randomly generated” and “randomly generated” starting points [49]. They find no evidence that “better than randomly selected” initial solutions lead to lower objective function values in the final solution (on average), prompting the researchers to conclude that a properly tuned SA algorithm can produce good results regardless of the initial solution. It is also seen in the literature that the number of iterations is another parameter to which the results are sensitive [51], as can be expected.

Some researchers design the algorithm such that a large number of solution updates is performed at every temperature value or within a temperature range [8, 49]. This number is yet another variable that is to be determined experimentally. Tan and Narasimhan test 1, 50 and 100 such updates per temperature range and find that increasing this number from 1 to 50 results in almost twice the number of optimum solutions being identified by the algorithm, but a further increase from 50 to 100 reduces this to 60% [49]. They then conclude that it would probably yield better results to use a smaller temperature decay function than a larger number of updates at each single temperature value. Raaymakers and Hoogeveen also find that the number of iterations with the same temperature does not affect the final solution quality [22]. We rely on the conclusions of these researchers and choose only to do one update at each temperature and use a log temperature decay function to ensure a gentle reduction. Saraiva et al. run experiments with 200 problem-generation loops per temperature [8] but do not investigate alternatives or different values to allow comparison.

The experiments of Jozefowska et al. are based on a pseudo-random number generator which is seeded with the same value at the start of every run [50]. This method may introduce bias and casts doubt on the generalisability of the results. The results are presented as a number of times out of the total number of runs in which an optimal solution is identified. For 10-job schedules, average relative deviation without penalty functions is 0.93% and sky-rockets to 110% maximum in 60 000 iterations, with performance generally improving for larger problem instances. This trend may suggest that the results we present in Chapter 5 could also be improved in larger problem instances. Some experimenters give only running times, which does not allow for objective comparisons [60], or do not have instances of ten jobs and, most significantly, do not optimise the same criteria as we do. Wu at al.'s experiments on SA with learning effect had results ranging from an average percentage deviation of 1% to a maximum of 1371% and even 3867% for 12-job instances [54]. Such big discrepancies indicate that more work is required on scheduling using SA to get more consistent results.

Many researchers use mean CPU run times of their algorithms as a performance measure (examples are [16, 52, 54]) but neglect to present the complexity order. This makes comparisons fairly meaningless as myriad factors affect running times, including the platform (hardware and Operating System), other software running on the platform, the language in which the algorithms are coded, and the developer's coding technique.

This is why we have opted not to record running times and only consider complexity order in our work.

3.2 Genetic Algorithms

GAs were developed by Jon Holland and his students, with the intent of investigating a more general method than had previously been developed to apply natural adaptation to computer systems [61]. The GA consists of a starting set of candidates - usually a subset of the search space. These are called the initial population chromosomes. An important consideration to be made in setting up a GA is the representation of the chromosome structure to be used, since this can affect the final result significantly. In the case of task scheduling, a chromosome is usually represented by a permutation of a job sequence, i.e. a schedule [5], while individual jobs characterised by their processing time represent genes [17, 62, 63] with additional constraints imposed where necessary. Batch [6, 16, 62] and multiprocessor scheduling [6], or scheduling with pre-emption allowed [16, 63], may require more exotic representations. The representation structure directs the application method of selection and the genetic operators.

A fitness function is used to quantify the performance or quality of candidate solutions, and rank and select candidates accordingly. Operations on chromosomes include crossover and random mutation to produce future generations. GA is differentiated from other search methods in that several solutions are generated and carried over at each iterative step of the algorithm.

Selection methods by which chromosomes are chosen to produce offspring are generally divided into roulette wheel, rank, steady-state and elitism. In roulette wheel, the proportion of an individual's fitness value to the average fitness value of the population is used as the expected value of the number of times that individual is chosen to reproduce. The actual selection for each new generation is performed statistically based on these expected values [61]. This method is prone to premature convergence as insufficient exploration is a common affliction. In elitism some number of the fittest individuals is always retained at each generation, without allowing the destruction of their genes by operations such as mutation or crossover. This is usually used in addition to other selection methods. In rank selection, individuals are ranked according to fitness. The expected value then depends on the rank rather than proportional or absolute fitness. Ranking reduces the overwhelming effect of large fitness differences on the selection, and so reduces the likelihood of premature convergence of a population. In steady-state selection, only a few of the least fit individuals are replaced at every generation by the mutation and crossover operators.

These background points are made here to highlight the level of complexity, both computationally and practically, involved in implementing the GA, which is to be contrasted with the comparatively simpler SA algorithm.

In the literature on single machine scheduling using GAs there is very little that considers the same scheduling criteria as we do in this work. Of the more closely related work is that of Hamidinia et al. who seek to minimise total weighted tardiness and earliness of jobs without pre-emption but in the case of a batched delivery system [16]. The authors also make the simplifying assumption of a due date as opposed to a due window as we do in this work. Other batch approaches are those of Zade and Fakhrzad who find the number of batches in which to divide the jobs, as well as the sequencing of jobs in the batches to enable maintenance to take place after a predetermined time period [62]. Other criteria considered in the literature are minimising completion time variance between completed jobs [18] along with the number of tardy jobs; minimum flow time and maximum earliness [63]; number of tardy jobs and maximum earliness assuming fixed due dates [17]; makespan with multiple heterogeneous processors [6, 62] within the constraint that the machine may be taken out of service for maintenance at certain times and with pre-emption allowed; as well as mixed criteria more closely tailored to the context [7]. Köksalan et al. aim to find all efficient schedules in the search space [63].

While many researchers start with a randomly selected population of chromosomes, some have employed certain heuristics to find a suitable starting point. For example, Köksalan and Burak Keha develop a heuristic that finds an approximately efficient solution set for a specific number of tardy jobs as the seed for the initial population [63]. This is also used by Jolai et al. [17]. An interesting method for generating neighbour solutions is employed, which follows from the representation of solutions as points in $[0, 1]^n$-space. Chromosomes are generated at certain Euclidian distances from the heuristic solution by first randomly generating a direction in $[-1, 1]^n$-space and then finding a vector that has the same generated direction and the required length. The new vector is added to the present solution and scaled appropriately to get tasks in the given direction each in the range $[0, 1]$. Since random number generation may be resource intensive, the authors use an "alias method" to reduce the number of random numbers to generate, to 1. However, the method requires a sorting algorithm to be run at least as many times as there are iterations, and entails many vector operations, both of which carry a major computational burden.

For the batch scheduling situation of Hamidinia et al., steady-state selection is employed with the top $\sqrt{n}$ individuals chosen to reproduce [16]. Mutation then occurs with 50% probability. Additionally, the number of mutation points depends on the number of consecutive generations with unchanged best fitness value in an effort to rescue the algorithm from local optima in a way that is in proportion to the severity of the situation. In [62], roulette wheel selection is used, with single-point crossover, while mutation frequency is also correlated to the number of iterations with an unchanged best solution. Partially matched crossover is used and mutation is implemented by swapping job positions, while a combination of crowding and elitism is used for replacement. All offspring with improved performance over the current population replace the equivalent number of chromosomes from the current generation, thus ensuring the best-performing chromosomes from the current generation are preserved. Crossover rate, mutation rate, population size and number of generations are included in the parameters investigated. In this work, mutation is carried out depending on the entropy of the population – when the entropy of a generation is below a certain threshold value – and repeated as many times as is required for the entropy to be raised to, or above, the threshold.

Roulette wheel selection is also used by Jolai et al. [17] and tournament selection is employed by Köksalan and Burak Keha [63], both with elitist strategy, while mutation occurs by choosing one chromosome and one of its genes randomly and changing the value of the chosen gene with 80% probability. Jolai et al. use a less effective fixed probability of 60% crossover and 40% probability of mutation [17]. In contrast, Gupta and Kumar experiment with 90% crossover and 5% mutation rate [18]. Since all the algorithms have differing criteria and problem sizes, it is not meaningful to compare results so this has been omitted by intent. The impression from this section is that GAs are highly computationally expensive and intricate to implement.

3.3 Tabu Search

Generating weighted earliness-tardiness problems similarly to the method we use in this work, Wan and Yen investigate the performance of Tabu search methods in solving the problem with task sizes of 15 to 80 jobs [30]. A weakness is that worst- or average-case computational complexity is not given mathematically; only running times are given, which does not allow for fair comparison. It is claimed that their search method usually obtains deviations within 5% of the optimal value, with an average 2% deviation for 15-job problems. The worst-case values or variances are not given, which may lead to misrepresentation of the actual performance. Hino and Ronconi address the earliness-tardiness problem using Tabu Search, but include heuristics for finding a suitable starting schedule [33]. They find deviations ranging from 0% to 0.25% from the benchmark (not the optimal) for ten-job problems, solving 216 of 280 instances optimally. The complexity of this algorithm is $O(n^2)$. Wodecki finds errors below 5% for 40-job earliness-tardiness problems after $2n^2$ iterations, but also uses another algorithm to find a suitable starting point [64]. Other related Tabu Search algorithms for scheduling are for the total tardiness problem [65], minimising makespan [64], number of tardy jobs and average flow time [66] and parallel machine scheduling [67].

3.4 Branch and Bound

The Branch and Bound (B&B) procedure is a suitable optimal algorithm for scheduling problems, more storage efficient than other optimal algorithms such as dynamic programming and, as such, has been used in a number of works that seek optimal schedules [4]. Yano et al. present a B&B algorithm as optimal with heuristics used to provide an upper bound to prune the search space [68], as do Ronconi and Kawamura [69] and Wan and Yen [59]. Yano and Kim determine that the optimal timing algorithm for the special case of equal weights is $O(n^2 \log n)$ [70]. Tan et al. find that B&B would be the preferred solution for smaller problems compared to SA and GAs since it yields an optimum solution in an acceptable time period [71], however we seek a solution that could be applicable to both small and large problems. Their particular problem also considers sequence-dependent set-up times in the minimisation of total tardiness. This algorithm obtains upper bounds by starting with depth-first branching, so that each sequence is traversed as far as possible before stopping or turning back. Mazdeh et al. compare the performance of a B&B algorithm with a Dynamic Programming (DP) method and show that using B&B gives significantly better efficiency than the DP method owing to the high time complexity of DP [60].

3.5 Dynamic Programming

The method of Dynamic Programming has received quite a lot of attention in recent years owing to its general nature that allows it to be used to optimise a wide variety of systems, including non-linear systems. A number of researchers have used DP in solving scheduling optimisation problems and, in particular, the single machine scheduling problem. See [27, 31, 72–77] for examples. DP does have its disadvantages, though. It is not the most efficient of optimisation methods. That is why the use of certain precedence or dominance properties of the problem to prune the solution search space is a recurring theme in DP research.

Yano and Kim's investigation of the single machine scheduling problem with weighted earliness and tardiness shows that for a specific subclass of the problem under specific conditions, rules of precedence can be found with a simple sorting algorithm [27]. This then provides the optimal sequence and so simplifies the process of obtaining the optimal solution. The optimal timing is then found easily using DP, on the assumption that a sequence is given. The conditions of the subclass of problems are similar to those used by Chand et al. [72]. The performance of the dynamic algorithm or the combination of sorting and timing algorithms are not evaluated. DP is used by Tanaka and Sato who propose an exact algorithm for the precedence-constrained single machine problem that is based on Sublimation Dynamic Programming [78]. A solution is found for the objective of minimising general additive costs instead of a specific restrictive function. They claim that their algorithms outperform all others up to that date (2013) on the same non-precedence-constrained problem set.

Shabtay also presents a DP-based optimisation algorithm for a bi-criteria batch scheduling problem with rejection, to minimise total completion time as well as the total rejection cost incurred when the scheduler is forced to reject jobs [79]. The author observes that the set of accepted jobs will have an optimal sequence where the SPT rule applies. This assumption is then used as the basis for the DP algorithm. The algorithm is proved to run in $O(n^5)$ time. It is thus very computationally expensive.

Ibaraki summarises Successive Sublimation DP (SSDP) by saying that it is a method that executes a series of Dynamic Programming recursions, refining or pruning the underlying state space as it continues. SSDP does this by eliminating states from the search space as soon as it is determined that they do not lead to the optimal solution [74]. The value that this SSDP algorithm adds over other algorithms of its time (such as B&B) is that its use is not restricted to non-decreasing objective functions, and so has a much more general field of application. State space relaxation methods are introduced for the algorithm and the percentage error computed and compared with the optimal solution. One relaxation method uses discrete time and introduces job penalties. For this case, the study shows percentage errors of the method's solutions as low as 0.23% in 85 iterations and up to 3.13% in 80 iterations, for a range of problems, with $N = 30$. They obtain still more accurate results when starting with upper bounds found by simple heuristic procedures, introducing the constraint of maintaining the linear order of jobs, and then solving the constrained version of the DP to improve the accuracy of the upper bound found by heuristic. In many cases, this method results in 0% error. In various simulation experiments, the running time of SSDP was found to be significantly lower than for the original DP, when the number of states (i.e. jobs) is above a certain value.

SSDP is also used by Tanaka et al. as a basis for their proposed algorithm [78]. In this version of SSDP, Lagrangean relaxation is applied and the relaxed problem solved to find the lower bound, and then constraints are added successively to this problem until the error bounds disappear. Their numerical experiments were carried out using problem instances from the open source OR-Library. Processing time, tardiness weight and due date were generated from a uniform distribution, and the CPU times were recorded for both DP and their SSDP solution.

3.6 Other hybrid or combined algorithms and approaches

Estimation of Distribution Algorithm (EDA) is a probabilistic algorithm with a complexity of $O(n^2)$. In [80], EDA is combined with GA to improve the effective complexity of EDA. The addition of genetic operators to EDA prevents over-fitting and premature convergence, while EDA is used to characterise parent solutions in the GA, so that favourable features may be maintained.

Yannibelli and Amandi combine SA and an evolutionary algorithm, where the multi-objective SA algorithm is integrated into a multi-objective evolutionary algorithm [81]. In their formulation, the stage of the evolutionary algorithm and level of diversity of the population changes the behaviour of SA. During the earlier iterations of the evolutionary algorithm, SA is used to reduce the now diverse population of solutions for fine tuning. Towards the later stages of the evolutionary algorithm, SA is used to increase the diversity and encourage exploration to prevent premature convergence. The problem addressed in this work is very different from ours, however, since they look at scheduling starting times of project activities and assigning the most effective human resources to those activities so as to minimise makespan.

Ali and Bijari develop a two-stage heuristic approach for minimising the sum of maximum earliness and tardiness [82], which is similar, but not the same, as our considered objective. The first step of their heuristic involves finding the approximate position of jobs in the sequence, and the second step refines the positions using an Hungarian algorithm. Of problem instances with ten jobs, their algorithm solved 18 of 20 problems optimally, with an average percentage deviation from the optimal of 0.28%. In another set of problems, 11 of 20 were solved optimally with an average deviation of 3.34%. Ji et al. find an $O(n^4)$ solution algorithm for the single machine scheduling problem with a common due window and rate-modifying activity, to find the minimum cost of earliness, tardiness, due window start time, size and resource consumption [83]. They also provide an $O(n^2 \cdot \log n)$ algorithm for the special case of identical rate of ageing for all jobs. Cheng and Cheng show that the algorithm of Ji et al. is incorrect, but remains valid for the case of a common modifying rate of 1 [84].

A greedy randomised adaptive search procedure is employed for the single machine scheduling problem for minimising total tardiness, with the added complication of sequence-dependent set-up times in [85]. Researchers have investigated other approximation algorithms [86] and memetic algorithms (which include scatter search) [87, 88], claimed to combine the strengths of a hierarchical population approach (such as in GA) and the "intensification power" of local search procedures [88]. Neighbourhood search [89, 90], particle swarm [91] and recovering-beam search [32] are all examples of alternative techniques that appear in the literature. Problems modelled as multiple competing agents are presented by Perez-Gonzalez and Framinan, who review existing solutions [92] and Mor and Mosheiov, who present a polynomial time solution for a specific two-agent problem [93]. The proposed Goal Programming method of Li, Fonseca and Chen promises to ensure a global optimum solution is found, but fails to highlight the trade-off of increased computational complexity [94]. Linear Programming solutions that include certain upper bounds are presented by Ng, Cheng and Kovyalov [95] and an analysis of a Linear Programming heuristic by Potts shows that for the minimisation of maximum completion time on two parallel machines, a linear-time algorithm can be used [58].

3.7 Metropolis-Hastings Markov Chain Monte Carlo

Markov Chain Monte Carlo methods have arguably been the most influential algorithms of the 20th century, being used in fields as varied as Physics, Engineering, Econometrics, Statistics and Computer Science. Surprisingly, this kind of Monte Carlo does not appear to have been used in scheduling applications at all, let alone the Metropolis-Hastings method. Developments are ongoing with improvements such as adaptive MCMC [96–99], Hamiltonian Monte Carlo [100, 101], reversible jump MCMC [102] and various papers on convergence diagnostics [46, 47, 103, 104].

3.8 In summary

This concludes our account of some of the work most relevant to ours. We see that most approaches to the problem are quite different from our own, with no previous work combining SA and Metropolis-Hastings MCMC in the way we have. We hope that this has highlighted some gaps and started to provide a justification for the work we present here. Further motivation will be provided in later chapters as appropriate.

Chapter 4

Simulated Annealing with Metropolis-Hastings pre-sampling

The details of our new hybrid algorithm, which we call SAM for Simulated Annealing with Metropolis-Hastings, are now presented. Briefly, in this algorithm a pre-sampling sequence is first performed to funnel in on one section in which further search may be done so that a solution to the single machine weighted earliness-tardiness problem can be found that is near-optimal (or possibly even optimal in some cases) in a way that is less computationally intensive than existing methods. The pre-sampling is done by Metropolis-Hastings MCMC to infer the section that is most likely to contain an optimal or low-deviation solution. Once a section has been chosen, the search space of possible schedules is greatly reduced and SA can be used more efficiently to find a solution.

4.1 Formulation

In this section we formulate and motivate how the problem is modelled as a Metropolis-Hastings (MH) sampling problem and how the interpretation leads to a more efficient search procedure. We start with some notation and definitions.

Figure 4.1 represents the evolution of the Markov Chain. The figure shows that states i are drawn from the set of section indices $\mathbf{I} = \{1, 2, \dots, m\}$ with the prior distribution on

$$\mathbf{I} \sim P(\mathbf{I}) : p(i) = \frac{1}{m} \quad \forall \quad i \in \mathbf{I} \quad . \tag{4.1}$$

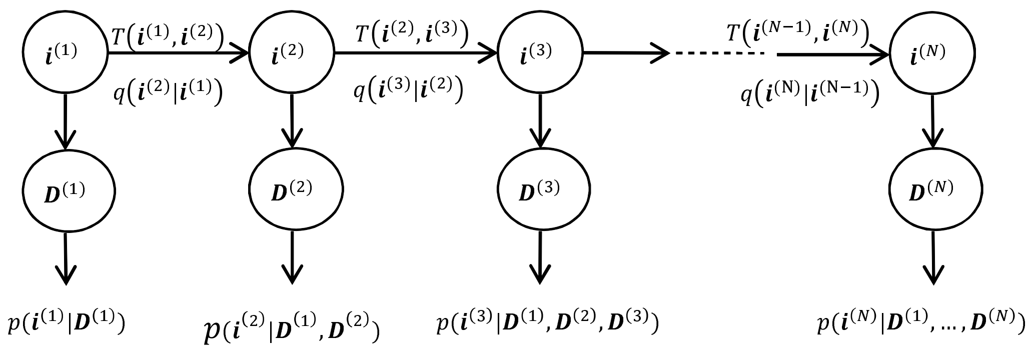


FIGURE 4.1: The evolution of the Markov Chain with sample data gathered on each state change

The set $\mathbf{I}$ divides the full set of schedule permutations $\mathbf{S}$ into sections 1 to m. Each index $i \in \mathbf{I}$ maps to the set of schedules $\mathbf{S}(i) \subset \mathbf{S}$ within a range i of size $\frac{|\mathbf{S}|}{m} = \frac{n!}{m}$ for an n-job schedule, as per eqn. (4.2):

$$i \to \mathbf{S}(i) \ = \ S_x, S_{x+1}, ..., S_{x+[\frac{n!}{m}]-1} \quad . \tag{4.2}$$

The target distribution is the posterior distribution of $\mathbf{I}$, which is inferred from data $\mathbf{D}$ gathered after every sample drawn from $\mathbf{I}$. We require a function to calculate the likelihood that a chosen section i contains an optimum schedule, which we can calculate from the cost of schedules in the chosen section. This likelihood is a representation of our belief that the optimal schedule lies within the range $\mathbf{S}(i)$. An example of a way to get a likelihood value of a specific section i from the costs is eqn. (4.3).

$$L = \frac{(\min\limits_{s \in \mathbf{S}(i)} \gamma(S_s))^{-1}}{\sum_{j \in \mathbf{I}} \min\limits_{x \in \mathbf{S}(j)} \gamma(S_x)^{-1}} \tag{4.3}$$

Instead of using all schedules in $\mathbf{S}(i)$, a value is estimated from cost values of a subset of schedules sampled in i i.e. the evidence dataset $\mathbf{D}_i \subset \mathbf{D}$, where $\mathbf{D}$ is all samples drawn from all sections thus far, as shown in eqn. (4.4).

$$L(\mathbf{D}|i) = \frac{\left(\min\limits_{s \in \mathbf{D}_i} \gamma(S_s)\right)^{-1}}{\sum_{\mathbf{D}} \left(\min\limits_{x \in \mathbf{D}} \gamma(S_x)\right)^{-1}} \tag{4.4}$$

We draw the evidence dataset $\mathbf{D}_i$ from a uniform distribution over $\mathbf{S}(i)$. We make the assumption of a uniform distribution as no further information is available about the statistical properties and shape of cost values in each section. This assumption is maintained throughout the procedure as it is not the goal in this pre-sampling process to define the nature of the distribution function within each section. The posterior $P(\mathbf{I}|\mathbf{D}) \propto L(\mathbf{D}|\mathbf{I})$ is updated every iteration and emerges from the MH procedure after a sufficient number of iterations.

The proposal distribution must be defined in a way that encourages wide exploration of the search space, and we must ensure that every section is sampled at least once so that the likelihood value can be updated with more realistic values. For this reason, we initialise the algorithm by drawing a sample from the prior $P(\mathbf{I})$ as the first state value. Data from schedules within the range of the sample state are gathered and the likelihood calculated based on the data. As part of initialisation, this process is repeated once in every section, setting $p(i|\mathbf{D}_i)$ as per the data for all $i \in \mathbf{I}$, and noting that

$$P(\mathbf{I}|\mathbf{D}) = \frac{L(\mathbf{D}|\mathbf{I}) \times P(\mathbf{I})}{P(\mathbf{D})}$$

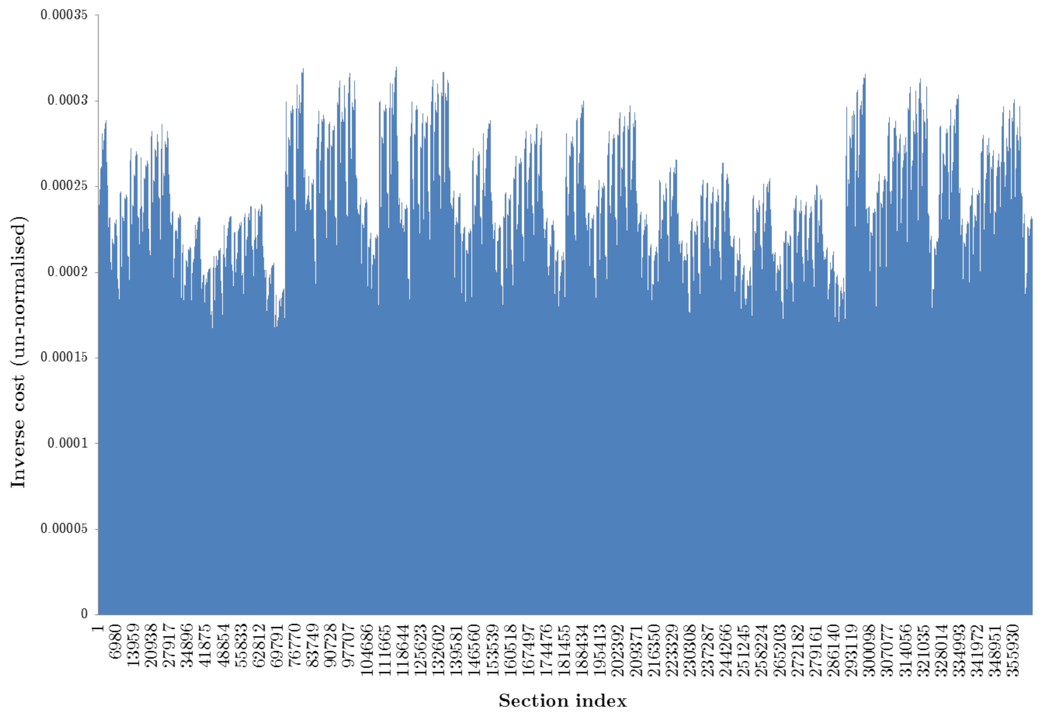
to scale appropriately. All subsequent samples are drawn from the conditional proposal distribution $q(i^{(t)}|i^{(t-1)})$, and transition to the next chosen candidate depends on the acceptance probability, eqn. (4.5):

$$\alpha = \frac{p(i^{(t)}|\mathbf{D}_i^{(t)})q(i^{(t)}|i^{(t-1)})}{p(i^{(t-1)}|\mathbf{D}_i^{(t)})q(i^{(t-1)}|i^{(t)})} \quad . \tag{4.5}$$

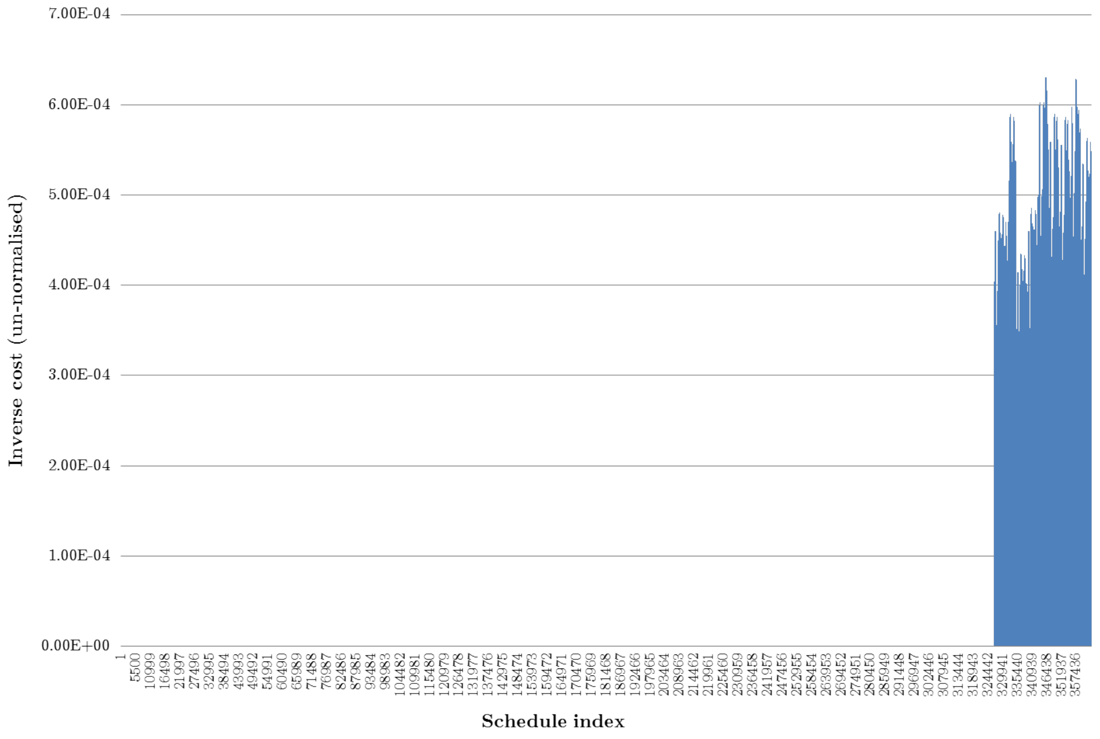
The value $p(i|\mathbf{D}_i)$ is updated every time a new sample is drawn from the set of sections, because the new samples change the denominator of the likelihood. This process continues until termination conditions are met, at which point a discrete piecewise-defined function has been quantified representing the likelihood over section indices of each section containing an optimal schedule. The section with maximum likelihood is selected, after which SA is performed within this section i.e. only considering schedules within the selected section.

SA is still useful for a fine search because it is able to start broadly according to the results of the rough search and then home in progressively on an optimum, which is now more likely to be a global optimum since the pre-sampling method has concentrated on *exploration* of the state space. SA then performs *exploitation* in a more fine-grained search on a smaller search space.

Having formulated our approach to the problem, we now provide motivation for some significant elements and assumptions.



(a) Dataset 5



(b) Dataset 8

FIGURE 4.2: Bar graph of the indices of the schedule permutations against the inverse cost of the corresponding schedules. A certain general clustering of similar cost values can be observed

4.2 Motivation for the section-based pre-sampling approach

The foundation for the solution methods to the single machine scheduling problem presented in this work is the way the problem is visualised. The search space $\mathbf{S}$ is the set of all permutations of the base schedule, with indices corresponding to schedule $S_x \mapsto x \in \{0, 1, 2, ..., n! - 1\}$ and the goal is to find the index corresponding to the permutation with the lowest cost γ, i.e. we seek the index of eqn. (4.6).

$$\min_{\mathbf{S}} \gamma(S_x) = \min_{\mathbf{S}} \left[\sum_{j \in J} (w'_j E_j(x) + w''_j T_j(x)) \right] \tag{4.6}$$

The search space is visualised as a discrete function in two-dimensional space with the horizontal axis as the set of indices of the schedule permutations sorted in lexicographical order and the vertical axis as a function which represents the likelihood that that schedule is the optimal and has the lowest cost. It is reasonable to assume that the likelihood of an index being optimal is inversely proportional to the cost, so the discrete likelihood function is assumed to take the form $c \cdot \frac{1}{\gamma(S_x)}$ over all S_x in $\mathbf{S}$ for c some normalising constant.

When the schedules are sorted into lexicographical order, observations of the cost over the x-space show that the values are naturally grouped or clustered roughly into sections in which the energy values fall within certain minimum and maximum energy limits. This behaviour is evident in plots for various different problem instances. It is inherent from the nature of the cost calculation and the sorting order, and is thus independent of the specific sets of processing times or penalties, although the weighting values may mask this behaviour somewhat. The example graphs of permutation indices with their corresponding cost values (Figures 4.2a and 4.2b) illustrate this clustering-like behaviour.

We claim that the range of total weighted earliness and tardiness (i.e. γ) *within* a section is, in general, less than the range of γ-values of schedules in *different* sections. We motivate by showing that the expectation of the difference in cost between schedules in the same section is less than the expected difference between costs of schedules in different sections, assuming specifically lexicographical ordering.

We proceed by noting that when job sequences are sorted lexicographically, for any number of consecutive sequences, there will be a fixed part and a variable part. The fixed part is the part of the sequence where certain job positions are fixed. For example, we consider consecutive sequences ordered lexicographically in A to D:

A: 1 2 3 4 5 6 7 8 9 10

B: 1 2 3 4 5 6 7 8 10 9

C: 1 2 3 4 5 6 7 9 8 10

D: 1 2 3 4 5 6 7 9 10 8

The set of sequences all have the positions of 1 2 3 4 5 6 7 fixed. Then sequences A and B also have the position of job 8 fixed, while sequences C and D have the position of 9 fixed. The cost γ of a schedule depends on the position of jobs, which determines how early or late each job is. If the variable part is varied from the end of the sequence all sequences sharing the fixed part will also have a fixed cost component. We can represent the cost as eqn. (4.7):

$$\gamma = c + \gamma_{var} = c + \gamma'_{var} + \gamma''_{var} \tag{4.7}$$

where we see the variable part of γ decomposed into the part influenced by earliness penalties (γ'_{var}) and the part influenced by tardiness penalties (γ''_{var}).

We can say there exist permutations $\Pi_1 = \{\alpha_1, \ldots, \alpha_n\}$ and $\Pi_2 = \{\beta_1, \ldots, \beta_n\}$ such that $\alpha_1 = \beta_1,\ \alpha_2 = \beta_2,\ \ldots,\ \alpha_{i-1} = \beta_{i-1}$ but $\alpha_i < \beta_i$ then $\Pi_1 < \Pi_2$. In this way the lexicographical ordering assigns permutation indices $N(\Pi)$ to permutations Π such that $N(\Pi_1) < N(\Pi_2)\ \forall\ \Pi_1 < \Pi_2$. The lexicographical metric [105] is defined as eqn. (4.8).

$$N(\Pi) = \sum_{k=1}^{n-1} (l_k - 1)(n-k)! + 1 \tag{4.8}$$

where l_k denotes the ordinal number of element α_k in the sequence. The distance between two permutations is $|N(\Pi_2) - N(\Pi_1)|$. For permutations 1 and 2 in the same section, and permutation 3 in a later section,

$$\begin{gathered} |N(\Pi_2) - N(\Pi_1)| < |N(\Pi_3) - N(\Pi_1)| \\ \sum_{k=1}^{n-1} (l_k(\Pi_2) - 1)(n-k)! < \sum_{k=1}^{n-1} (l_k(\Pi_3) - 1)(n-k)! \quad . \end{gathered} \tag{4.9}$$

We know $\Pi_1 = \Pi_2 = \Pi_3\ \forall\ \{l_k = 1,\ \ldots,\ i-1\}$ and $\Pi_2 = \Pi_3\ \forall\ \{l_k = 1; \ldots; j-1\}$ for $j < i$. In each permutation, the ordinal number of i increases and n decreases so Π_1 has $i, \ldots, n-1$ the latest and Π_3 has $i, \ldots, n-1$ the earliest. This means

$$\begin{gathered} C_i(\Pi_1) > C_i(\Pi_2) > C_i(\Pi_3) \\ \therefore C_i(\Pi_3) - C_i(\Pi_1) > C_i(\Pi_2) - C_i(\Pi_1) \quad . \end{gathered} \tag{4.10}$$

For every job position shift later, another job must move earlier. However, we generate earliness and tardiness weights as per Section 5.1.1 so that on average $w' = 0.5w''$. This means that tardiness dominates the cost calculation over earliness and, on average, the increase in weighted C_i is greater than the decrease in weighted C_i when jobs are shifted in position, which is related to the cost since

$$E = \max(e_j - C_j;\ 0) \quad and \quad T = \max(d_j - C_j;\ 0) \quad .$$

In a section, the maximum expected increase in C_j can only be within average processing time multiplied by the maximum position change $j(in - section)$, which is less than $j(between - section)$,

$$\therefore \mathbb{E}[\Delta C_j(in - section)] < \mathbb{E}[\Delta C_j(between - section)] \quad .$$

γ_{var} is directly related to the number of job position changes j. As observed, both the maximum and expected number of position changes within a section are smaller than the maximum and expected number of changes in the whole search space, and so

$$\begin{gathered} \max j(in - section) < \max j(between - section) \\ \therefore \mathbb{E}|\gamma_{var}(in - section)| < \mathbb{E}|\gamma_{var}(between - section)| \end{gathered} \tag{4.11}$$

since $\gamma''_{var} > \gamma'_{var}$.

In the example, all four sequences will have equal c for jobs up to 7 while A and B will further share the same cost incurred by job 8 and sequences B and C will share the cost component of job 9. Therefore, consecutive schedules A and B will have a larger equal cost component c in common with each other than with sequence D, which is not consecutive and further away; and similarly for C and D vs. A and B. The greater the number of adjacent fixed positions between sequences, the greater the fixed part of the cost and the smaller the variable part of the cost is.

In general, this means that the cost values of sequences that share a large fixed component will be more similar than the cost values of sequences that have fewer job positions in common. Since schedules are sorted lexicographically, with job position changes occurring from the end, job sequences in the same section will generally have a larger fixed component and therefore will have similar costs compared to schedules in another non-adjacent section that would necessarily have a larger variable component, while they may have a larger fixed portion within the section.

From this, we conclude that, on average, the plot of the cost values of schedules sorted lexicographically will display a certain structure.

□

To illustrate the relative ranges of cost values between and within sections, we start by looking at the maximum difference within a section.

Suppose a section has width 36288, such that schedule 1 is 1 2 3 4 5 6 7 8 9 10. Using the C++ sorting algorithm, the final schedule in the section is 1 2 10 4 6 5 9 8 7 3. The maximum position changes are jobs 10 and job 3, both moving 7 positions. Job 10 moves to position 3 and job 3 moves to position 10. We shall call these the critical jobs.

The maximum cost to the schedule 1 owing to job 10 in position 10 occurs if it has the highest tardiness penalty, the earliest possible due date and the longest possible processing time. This is because tardiness weights are always larger than earliness weights using the problem generation technique in Section 5.1.1. Without loss of generality, let $RDD = T = 0.2$ for illustration. Let $p_{10} = 100$, and for the latest completion time, let $p_j = 100 \; \forall \; j \in \{1, ..., 10\}$. Let job 10 have the maximum tardiness weight $w''_{10} = 10$. Maximum tardiness penalty occurs when the job has the earliest possible end of due window, d_{10} given the above, so earliest d_{10} occurs when it is assigned the earliest possible centre of due window and the smallest possible window size as in eqn. (4.12).

$$d_{10(earliest)} = (1 - T - \frac{RDD}{2})C_T + 0.5 \times 1 = (1 - 0.2 - 0.1) \times 1000 + 0.5 = 700.5 \quad (4.12)$$

The cost of job 10 at position 10 is therefore given by eqn. (4.13).

$$\gamma_{10}(\Pi(1)) = \max\{C_j - d_j; 0\} \times w''_{10} = (1000 - 700.5) \times 10 = 2995 \quad (4.13)$$

Job 3 has its maximum cost in the first schedule at position 3 if it is assigned the latest due window and the maximum earliness penalty. This makes $e_3 = 999.5$, $d_3 = 1000.5$ and $w'_3 = 9.9$. The maximum cost of job 3 at position 3 is then

$$\gamma_3(\Pi(1)) = \max\{e_3 - C_3; 0\} \times w'_3 = 6925.05 \quad . \quad (4.14)$$

Now suppose that job 10 is moved to position 3 as in schedule 36288: 1 2 10 4 6 5 9 8 7 3. No further properties of the job are changed so it is now scheduled too early. The start of the job's due window is

$$e_{10}(1 - T - \frac{RDD}{2})C_T - 0.5 \times 1 = 699.5 \quad (4.15)$$

and it is now completed at time 300.

The cost of job 10 at position 3 with maximum earliness weight, $w'_{10} = 9.9$ is now

$$\gamma_{10}(\Pi(36288)) = \max\{e_{10} - C_{10}; 0\} \times w'_{10} = 3955.05 \quad . \quad (4.16)$$

Job 3 is moved to position 10, now having a completion time $C_3 = 1000$. This is within its due window so it incurs no penalty.

All other jobs in schedules 1 and 362288 have moved fewer positions than jobs 10 and 3. Since job 10 was given the greatest tardiness weight and actual lateness, and job 3 was given the greatest earliness penalty and actual earliness, the cost difference for any other job must therefore be lower and these moves are less significant to the total variance on average.

The maximum difference *within* the first section is given by

$$\begin{aligned} \Delta\gamma(within - section) &= \gamma(\Pi(1)) - \gamma(\Pi(36288)) \\ &= 2995 + 6925.05 - 3955.05 \\ &= 5965 \quad . \end{aligned} \tag{4.17}$$

Now the maximum difference *between* sections is dependent on the maximum number of position shifts possible, i.e. 9, shifted by job 10 forward, and by job 1 to the end. We have said that job 10 has the highest tardiness penalty, the earliest possible due date and the longest possible processing time, with $p_j = 100 \ \forall \ j \in \{1, ..., 10\}$. Job 10 still has due window start at $e_{10} = 699.5$, but completion time $C_{10} = 100$. The cost it incurs at position 1 is given by

$$\gamma_{10}(\Pi(3628800)) = max\{e_{10} - C_{10}; 0\} \times w'_{10} = 5935.05 \quad . \tag{4.18}$$

Job 1 incurs maximum penalty at position 1 if it has the latest due date and largest earliness penalty and is completed at $C_1 = 100$. This means the cost incurred is $\gamma_1(\Pi(1)) = 8905.05$. When job 1 is moved to position 10, it incurs no penalties as it is within its due window. The maximum that job 3 can move is from position 3 to position 9 since position 10 is occupied. It is completed at time 900 and still has $d_3 = 999.5$. This makes the cost incurred with maximum tardiness penalty 995.

The maximum difference between sections is as per eqn. (4.19).

$$\begin{aligned} \Delta\gamma(between - section) &= \gamma(\Pi(1)) - \gamma(\Pi(3628800)) \\ &\approx 18780.55 - 6930.05 \\ &= 11850.55 > \gamma(within - section) \end{aligned} \tag{4.19}$$

We conclude that the maximum difference between sections is greater than that within sections, so the range of costs between sections is greater than that within sections.

The sorting algorithm sorts the vector by increasing values. This means that values are not in strictly increasing distance from the first schedule in terms of a metric such as the Lee distance. Should a different algorithm be used to generate the permutations, such as Steinhaus–Johnson–Trotter [106] or Heap's algorithms [107], consecutive schedules are likely to be more reflective of a monotonically increasing distance metric, making relative differences in cost values within and between sections clearer. An evaluation of the algorithm we develop when used on schedules permuted by these and other algorithms is left as future work. Please note, however, that our method is, in fact, a general one and can be used for any objective function and any kind of cost value distribution. As will be seen, it is likely to be less accurate for a more uniform cost distribution but it is still applicable in any case.

For this reason, as well as the obvious largely statistically varying nature of all quantities, we claim clustering in this work in a general sense, not an absolute or exact sense. In general, in a new section, the critical job will move another position to the left, usually creating an even greater difference in cost, since tardiness overwhelms earliness, and other moves in general have less impact on the cost. This shift of the critical job creates a new norm around which the schedule costs cluster. This claim is made for the general case and shows the usual trend to be expected. It does not apply to every schedule, since the problem is NP-hard.

This behaviour can be exploited to assist in finding the global optimum more efficiently. The x-space is large for an SA algorithm to traverse and still poses a threat of insufficient exploration causing the algorithm to settle at a local optimum and miss the global optimum. For this reason, we devise a pre-sampling method to narrow the search space. To this end, the x-space is sectionalised into a number of intervals equally dividing the total space. Looking at a cost vs. index graph, a rough function of the sections vs. an average cost over that section is revealed. Since the likelihood must be proportional to the inverse of cost, a rough function is similarly assumed to exist for the likelihood function and, by extension, the probability distribution. The task of pre-sampling is to determine the section in which the optimum schedule is most likely to be found so that SA can be run within the limited search space of that section to find the solution with greater efficiency than without the knowledge of the section. The pre-sampling method ideally should have the following attributes:

i) Allow the estimation of unknown and complex functions from relatively few samples.

ii) Take advantage of the inherent structure of the index-cost function.

iii) Traverse large search spaces to estimate the function.

iv) Yield the same answer regardless of the starting point.

v) Work without knowledge of the normalising factor that converts the likelihood function $\frac{1}{\gamma}$ to a probability.

Metropolis-Hastings MCMC meets these criteria and, as such, has been chosen for pre-sampling in the novel hybrid algorithm presented here. Here, the Markov state is given a special meaning which is not just the index of the schedule but, most importantly, the section to which it belongs.

The full set of schedule permutation indices is subdivided into intervals. The sampling activity consists of selecting a section and then choosing indices within that section so that a concrete cost value can be used in the MH acceptance probability calculation. Every time an interval is sampled, "inner" sampling is done per interval. No data structures are required to be constructed for the search in this case. The relevant section in the range 0 to $m-1$ is simply determined by the floor of the $\frac{index}{|section|}$ where index numbering begins from 0, and $|section|$ denotes section size. An array of limits is kept and the bounds looked up. If a search algorithm were required, the minimum complexity would be lower bounded at $O(\log n)$. In this case it has been kept to a constant, excluding the array storage. This method of uniform sampling is a constant time operation where the number of samples taken is some ratio of the number of permutations of the n jobs. The question is whether the search space can be reduced effectively by its employment.

The only knowledge we have of the actual distribution π is the assumption by intuitive understanding that it must be proportional to $\frac{1}{\gamma_{S_i}}$ but the constant of proportionality is unknown. This is the classical situation for the use of Metropolis-Hastings. The difference is only that, not only can the real distribution not be defined, but the target distribution for the MH algorithm is discrete and cannot be defined either, but a value proportional to it can be calculated for each sample. Samples can be drawn from a proposal distribution and accepted according to the MH acceptance probability so that a Markov Chain is constructed that fulfils the conditions of ergodicity and stationarity with respect to the target distribution. As long as the sampling procedure of MH is followed, these conditions are met and the Markov Chain will eventually converge to the desired distribution [39].

At inception, there is no *a priori* knowledge of the likelihood of sections, so all sections are given uniform likelihood. The likelihood values of the sections are then built up and refined as the number of samples drawn increases and more knowledge of the relative costs is gained. After a sufficient number of iterations, the chain of samples is assumed to have mixed to such an extent that a histogram of the accumulated samples resembles the actual probability distribution. Whereas in most applications, the bins used for the histogram are made small enough to give a fine view of the probability distribution, the form of the histogram we generate is peculiar in that the search space is not divided into the smallest possible bins, but into the bins as large as the sections, since the goal is to get a rough idea of the way in which the cost values cluster in certain areas. The section that boasts the highest likelihood is then chosen as the reduced state space in which SA performs a fine search.

It has been motivated why this pre-sampling approach is reasonable and likely to yield improved results. We now give further details of the algorithm.

4.3 SAM

We call the novel combination of Metropolis-Hastings pre-sampling with Simulated Annealing on the reduced search space, SAM. The SAM algorithm consists of three main phases: initialisation, MH sampling and SA. The full algorithm is illustrated in Figure 4.3 and the initialisation procedure in Figure 4.4 (both are placed at the end of the Chapter). Each step is discussed in the subsections below.

4.3.1 Initialisation

In the initialisation phase, the input parameter values to the algorithm are set. The set of sections indexed $\mathbf{I} = \{1, 2, \ldots, m\}$ has a certain p.d.f indicating, for each $i \in \mathbf{I}$, the likelihood that the overall optimal schedule s^* is found in that section. On initialisation an *a priori* p.d.f. $P(\mathbf{I})$ must be defined for the set of sections, which represents our initial beliefs about the shape of the p.d.f. The starting assumption is that $P(\mathbf{I})$ is a uniform distribution. The starting value for MH is a sample section index drawn from $P(\mathbf{I})$. We illustrate this in Figure 4.4.

4.3.2 Metropolis-Hastings: implementation details

Our specific implementation of the MH algorithm is shown in Algorithm 1. For each iteration of MH, a sample is drawn from $P(\mathbf{I})$. A quick search is performed within the section and from these results a likelihood value is deduced, the acceptance probability calculated and the next sample chosen.

This section provides further clarification of Algorithm 1. MH has been implemented in an object-oriented manner, allowing the user to pass in essential arguments and parameters that affect the algorithm's performance and facilitating easy expansion to

Algorithm 1: Metropolis-Hastings algorithm pseudo-code

```
Input: N, x, r, burnin, numSections, start, varianceFactor, randomEngine,
       samples[ ], vector<int> , accSamples
Output: The largest element in the set
Define: proposalpdf, targetpdf, uniform_real_pdf, GewekeTest()
forall $i \in$ burnin do
    candidate $\leftarrow$ draw sample from proposalpdf(x)
    draw r samples from uniformpdf(candidate)
    foreach inner sample $\in$ candidate do
        calculate cost
    calculate likelihood of candidate
    calculate $\alpha$
    draw uniform random value $u$ in (0, 1)
    if $u \leq min(\alpha, 1)$ then
        $x \leftarrow$ candidate
        update proposalpdf(x)
        $a \leftarrow 1$
    else
        $x \leftarrow x$
        $a \leftarrow 0$
while terminating conditions not met do
    forall $i \in N$ do
        candidate $\leftarrow$ draw sample from proposalpdf(x)
        draw r inner samples from uniformpdf(candidate)
        foreach inner sample $\in$ candidate do
            calculate cost
        calculate likelihood of candidate
        calculate $\alpha$
        draw uniform random value $u$ from (0, 1)
        if $u \leq \min(\alpha, 1)$ then
            $x \leftarrow$ candidate
            accSamples $\leftarrow x$
            update proposalpdf(x)
            $a \leftarrow 1$
        else
            $x \leftarrow x$
            $a \leftarrow 0$
acrrate $\leftarrow |accSamples|/N$
if GewekeTest() returns true then
    assume suitable convergence to correct value
    Select winning section
```

include new distribution types without exposing or needing to alter the core functionality. The implementation of generating and seeding the distributions is hidden, as well as the routine itself, the stored samples and current state of the algorithm. In the initialisation phase, the parameters passed in from the main program are: number of iterations (i.e. samples) (N), burn-in sample size (`burnin`), chain start point (`start`), the number of sections in which the search space is to be divided (`numSections`), and the number of jobs in the problem instance (`numJobs`). In addition, a variable (`varianceFactor`) can be passed. The significance of this factor is clarified later in this chapter.

Typically, in MH implementations very little guidance on the choice of proposal distribution is available and this aspect is left open to the engineer. A number of different proposal distributions were tested, more detail of which is given below but the symmetric random walk proved to yield the most consistent results. Because there is essentially no knowledge available to the system before the start of the algorithm, and to avoid the introduction of any bias, the most generic proposal distribution – the uniform distribution – is chosen and is probably the most successful for those reasons of limiting bias in case the algorithm is not run for long enough. Since the algorithm cannot be run to infinity, a certain degree of bias cannot be eliminated completely from the resulting solution. It is our claim that using a uniform distribution for the proposal distribution reduces the bias that could originate from the proposal. To take advantage of the Markov dependence of samples and progressively move towards a specific section, however, the option is added to define the distribution over a moving window that depends on the current state of the algorithm. The variance factor variable (herein denoted k) is introduced to give some flexibility in the implementation of the size of the window from which samples are to be drawn every iteration i, the distribution being defined in the interval

$$\left[x^{(i-1)} - \frac{N!}{M} \times k,\ x^{(i-1)} + \frac{N!}{M} \times k\right] \quad .$$

The size of the variance factor k controls the extent to which the proposal distribution retrieves samples from other sections, controlling the spread. In future work, this factor could be made adaptive, becoming larger when acceptance rates show that more exploration is required, and smaller when more exploitation is preferred.

It is not immediately obvious that this restriction on the proposal distribution state space maintains the conditions for irreducibility of the final distribution. This is addressed in [39]. Besag proves that it is not necessary for each transition kernel to maintain irreducibility individually but only that the combination of all transition kernels achieve irreducibility. He shows that if a single transition probability for state i is T_i, which only allows transition within some subset of the state space $\mathbf{S}(i) \subset \mathbf{S}$, some combination of T_i such as $T = T_1 T_2 ... T_N$ maintains the possible transitions of all the component kernels. Also, if T_i individually maintain the stationary distribution then the combination of transition kernels do so as well. In constructing the proposal distribution it is thus only necessary for the combination to allow transition to any other state so that the chain will have the property of irreducibility and can converge to the desired stationary distribution. At any rate constructing transition kernels from uniform distributions in this way still ensures that general balance (eqn. (2.25)) is satisfied. The method is, therefore, validated.

At iteration i, we choose a sample $x^{(i)}$ from a section m within the set of sections $\mathbf{I}$ with size M, which maps to a subset of the search space $\mathbf{S}$ i.e.

$$x^{(i)}(I_m) \in I_m \to S(I_m) : S(I_m) \subset \mathbf{S} \quad .$$

If we call the start point $x^{(0)}(I_m)$, we choose the next sample such that $x^{(1)}(I_l) : l \neq m$ to ensure exploration of the search space. We may continue imposing this condition until every section is represented by a set of inner samples before continuing the algorithm without the restriction. Since the proposal distribution is symmetric, the Metropolis-Hastings acceptance ratio simplifies to the Metropolis acceptance probability:

$$\alpha = \min\left\{1, \frac{\pi(x^{(i)})}{\pi(x^{(i-1)})}\right\} = \min\left\{1, \frac{k \times \frac{1}{\gamma(x^{(i)})}}{k \times \frac{1}{\gamma(x^{(i-1)})}}\right\} = \min\left\{1, \frac{\gamma(x^{(i-1)})}{\gamma(x^{(i)})}\right\} \quad . \qquad (4.20)$$

This value is compared to a uniformly generated random number u in $(0, 1)$ and, if u is less than α, the point is accepted and the cost added to an accumulator.

4.3.3 Simulated Annealing

Neighbour Generation

As mentioned, neighbour generation is a strategic part of the implementation, having a definite influence on the performance [8], yet in the literature shift, swap and insertion operators are used almost exclusively. These methods add to the algorithm run time and storage requirements.

In our representation, different solutions are indices to different permutations of the job set (different possible schedules). This implementation circumvents the usual computational load of neighbour generation and reduces memory requirements by visualising the search space as a set of permutation indices 0 to $n! - 1 (= 3628799)$ arranged radially similarly to the hours of a clock, as shown in Figure 4.5. Instead of a schedule being represented by a ten-job set in a certain order, the algorithm sees only indices that represent the permutations. The permutations, which would need to be stored as arrays, do not need to be stored at all. The actual schedule at a specific index is only called to calculate the cost if that schedule is sampled.

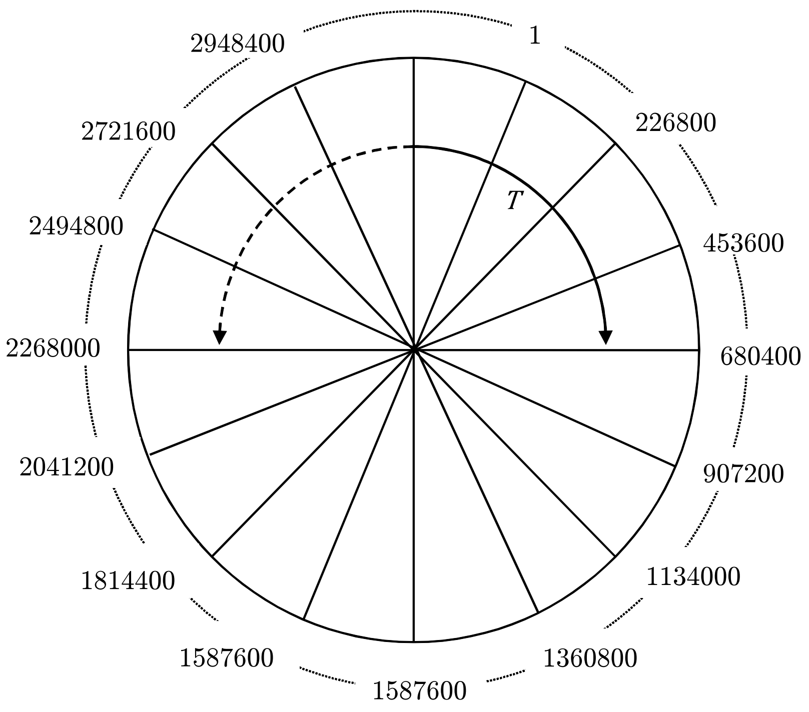


FIGURE 4.5: Representation of the set of permutation indices as a wheel with temperature parameter specifying how far around the circumference the algorithm traverses.

In Simulated Annealing the temperature is related to the extent of perturbation of one solution to obtain another neighbour solution. The temperature perturbation gives the range around the current schedule index of the distribution from which the next schedule may be chosen. The neighbour solution is selected from an integer uniform distribution T indices above and T indices below the current schedule i.e. for a schedule x, that would be $(x - T,\ x + T) \subset \mathbb{Z}$, a range represented by the arrows in Figure 4.5. Visualising the search space in this circular way eliminates the problem of "falling off the edge" of the search space by a temperature that causes a perturbation beyond the maximum index or a negative index. Such values will result in the algorithm simply counting forward or backwards respectively from 0 again. Eliminating the need to truncate the temperature distribution strengthens the validity of the sampling method by eliminating bias and what would have been a reduction in the randomness of the distribution. This also enables us to use different distributions such as Gaussians without synthetically truncating the distribution. The sampler simply walks around and back to the beginning.

Only to smooth the process of performing the experiments, all permutations of the set of jobs $\{1, 2, 3, ..., 10\}$ were found, sorted lexicographically and stored in an indexed array. This set of permutations need only be generated once on initialisation of the MH algorithm and is reused for each run of SA and indeed for any simulation of ten jobs regardless of the weightings, job lengths or any other specific feature of the problem instance. A perturbation algorithm has thus been reduced merely to a lookup. This unique way of visualising the search space has enabled a simplified neighbour generation method.

Parameters

SA is a very useful tool to use in finding good suboptimal solutions to scheduling problems. It enables relatively good answers to be obtained without requiring in-depth knowledge of the details of the problem instance, such as the relative distribution of earliness and tardiness penalties, or variance among the task lengths or distribution of energies. This is what is required as we seek a tool that is as general as possible to apply to any problem instances that can occur. An observed limitation in the use of SA, however, is the variation of solution quality and running time depending on the parameters used. The starting temperature has been said to have an effect on how well the algorithm performs, in terms of how many iterations are required before a solution close to the actual optimal is found [49, 67, 108].

Theoretically, the initial temperature should ensure that a move to any other solution in the search space is possible. A starting temperature that is too small results in only a limited exploration of the solution search space unless, and possibly even if, the number of iterations is extremely large. A too-large starting temperature results in very loose solution estimates or, once again, a high number of iterations is required for the temperature to reduce to a small enough value, or pseudo-converge, and for the algorithm to settle on a solution. It is worth noting that the temperature can be seen informally as a measure of the dissimilarity between successive candidate solutions. If the temperature is still large by the final iteration, there is still high uncertainty in the solution and, therefore, it may be a solution of low quality and high percentage deviation that has been found. Since temperature is such a critical parameter in the model, it would be of great use to have a way to find a starting temperature that enables sufficient exploration of the terrain but that allows a sufficient narrowing towards a solution within a set number of iterations. Such a way would be yet more useful if it did not rely on an assumption that the temperature must reduce to a certain value in a set number of iterations, and instead is calculated on a probabilistic basis. Some attempts have been made in the literature to calculate a proper starting temperature value, but few using the temperature cooling function used in this work, and fewer that are generally applicable.

SA consists of a series of probabilistic selections of new solutions based on the temperature, by the Boltzmann distribution $e^{-(\Delta E/k_B T)}$. The Boltzmann constant, k_B is often omitted in non-physical applications, as we do. Although it is common for the temperature to decay by a constant cooling factor according to a scheme where $T_{i+1} = r\,T_i$ for

$\{0 < r < 1 \ : \ r \in \mathbb{R}\}$ [22, 53, 67], in the implementation of this work, the temperature values follow a logarithmically decaying cooling function. This method, also found to a lesser extent in the literature [109], allows finer changes according to the conclusions of Tan that slower decay rates are more successful than repeated iterations at unchanged temperatures [49]. The temperature at iteration $i > 1$ is given by $T_i = \lceil \frac{T_1}{\ln(i+1)} \rceil$. The added 1 in the denominator is to ensure that the temperatures are only decreasing, requiring the integer argument of $\ln(\cdot)$ to be greater than 2. We round up as indices must be whole numbers.

If we represent the initial temperature as T_1 and the energy (cost) of the schedule at state i as E_i, the probability of transitioning to candidate state i^* from i entails two possible events, either the cost of the candidate schedule is lower than the previous schedule, or the Boltzmann probability of the transition is less than a randomly selected number from a uniform distribution in the interval $(0,\ 1)$, as expressed in eqn. (4.21).

$$[E_{i^*} < E_i] \ \vee \ \left[q(T_{i^*}|T_i) = e^{-\left[\frac{E_{i^*} - E_i}{\frac{T_1}{\ln(i+1)}}\right]} < U(0,1) \right] \tag{4.21}$$

At each iteration, the probability of acceptance involves these two probabilities, so if one was to calculate this analytically for the entire chain of N iterations (intersection of all probable outcomes), this would require calculating

$$p(T_1, N) = \prod_{i=1}^{N} \left[p(E_{i^*} \geq E_i) \exp\left(-\frac{E_{i^*} - E_i}{\frac{T_1}{\ln(i+1)}} \right) + p(E_{i^*} < E_i) \right] \quad . \tag{4.22}$$

Suppose that the invariant distribution reduces to the form of eqn. (4.23) (which it does [109]):

$$Z_p \prod_{i=1}^{N} \exp\left[-\frac{E_{i^*} - E_i}{\frac{T_1}{\ln(i+1)}} \right] \tag{4.23}$$

If we wish to find an optimal value of the starting temperature, the Expectation Maximisation method may appear to be appropriate. Using the log likelihood reduction and ignoring, for now, the constant offset term, the problem is equivalent to

$$-\sum_{i=1}^{N} \left[\frac{E_{i^*} - E_i}{\frac{T_1}{\ln(i+1)}} \right] = -\frac{N}{T_1} \sum_{i=1}^{N} \left[(E_{i^*} - E_i) \ln(i+1) \right] \quad . \tag{4.24}$$

Three reasons why the analytical approach to calculating a suitable starting temperature is inappropriate and invalid are now presented.

1. To determine the optimum temperature using maximisation techniques would require finding a solution to

$$-\sum_{i=1}^{N}\frac{d}{dT_1}\left[\frac{1}{T_1}(E_{i^*}-E_i)\ln(i+1)\right]=\frac{1}{T_1^2}\sum_{i=1}^{N}(E_{i^*}-E_i)\ln(i+1)=0 \quad .$$

   The implicit assumption in using this method, however, is that the change in energy is independent of the starting temperature and only depends on the iteration count i. The Markovian dependence nature of the process invalidates this assumption, unless i is sufficiently large. We may replace i with N and perform the calculation assuming that the final iteration count N is sufficiently large and the first-order Markov Chain assumption holds, yet without running and observing the algorithm, "sufficiently large" is completely abstract.

2. It is very difficult, if not impossible, to characterise $p(E_{i^*} < E_i)$ accurately. If this were possible, in theory it would be possible to jump to the optimal solution with relative ease.

3. Even if the second term were eliminated by shifting it to a certain constant factor in the Markov Chain's invariant distribution (as in [109]), the inherent limitation remains. Analytical methods to determine speed of convergence, or the number of iterations required to ensure that the solution found is outside the optimal set with a probability less than a certain ϵ, are largely useless. Any analytical method involves the calculation of asymptotic tendencies and is irrelevant to practical implementations [109]. Such methods intrinsically are not suitable even to prove superiority of SA over exhaustive search. The calculation required to ensure probability of $1-\epsilon$ of finding a solution in the optimal set S^* after N iterations, given starting T and cooling schedule, is equivalent to finding the starting T that ensures this probability in a given N iterations. The change of variable is a matter of Bayesian inference. If we could find N that solves $p(x_N \notin S^*, N|T_1) < \varepsilon$, the same method could be used to find T_1 that solves this for a given N since $p(x_N \notin S^*, N|T_1) < \epsilon \propto p(x_N \notin S^*, T_1|N)$. We can conclude, therefore, that the same limitations exist similarly inherently to the problem at hand of finding a "good" (optimal or close suboptimal) starting temperature.

Since it is concluded that analytical probabilistic methods are unsuitable to solving the problem of finding a "good" starting temperature, we turn to computational methods. The tested values have been based roughly on the relations between starting temperature and maximum iteration count shown in eqn. (4.25). The relation is based on the assumption that by the final iteration N, the temperature must have reduced to a value no greater than one, as this is the value of desired accuracy of the result, i.e. the solution should be within one schedule index accurate.

$$\begin{aligned} T_N = 1 &= \frac{T_1}{\ln(N+2)} \\ T_1 &= \ln(N+2) \\ \therefore N &= e^{T_1} - 2 \end{aligned} \tag{4.25}$$

As has been implied, the number of iterations is as much an integral parameter as temperature and indeed they are so interrelated, as is clear from eqn. (4.25), that in order to investigate the temperature variable without the assumption of final temperature of one, it is necessary that the number of iterations be fixed to an acceptable number. Preliminary runs showed larger temperatures – those a significant fraction of the size of the search space – are favourable. Owing to the complexity of performing this search and because it is not the main substance of the work, we have elected to investigate empirically the effect of only two starting temperature parameter values. The results are presented in Chapter 5.

The other influential parameter is starting point. This value we have fixed to the median index of the search space for all experiments. Since at the beginning of the SA journey of exploration, we have no knowledge whatsoever of the terrain, we must find a way to determine a good starting temperature that is effectively independent of starting point. Different combinations of parameter values were investigated and the results are discussed in depth in Chapter 5.

Computational Complexity of SA

In order to determine what factors have the greatest influence on the computational complexity of any modifications or other methods used to narrow the search space, the computational complexity of the base case SA algorithm implementation for this problem is analysed.

Theorem 4.1. *The computational complexity of the Simulated Annealing algorithm is $O(nN)$*

Proof. In each iterative loop of SA, the operations and their dependencies in Table 4.1 are executed (all k_l are constants).

No distinction is made here in the running time of random number generation and simpler mathematical operations such as additions, subtractions, multiplications and assignment. These are constants and irrelevant to the rate of growth. The starting temperature T_1, number of jobs n and number of iterations N are inputs to the algorithm.

Table 4.1: Calculating computational complexity of components of the SA algorithm

Compute new temperature	$\frac{T_{start}}{ln(i+2)} \approx k_1$
Find new solution Create new normal distribution and choose rand no. Shift solution placeholder to new solution and append energy to vector of energies. Compare energies	k_2
Calculate Energy $n-1$ times Find next entry in lengths vector Compute earliness Compute tardiness Compute sum	$O(n)$
Accept point? Compare energies negative OR Generate random number Compute Boltzmann value Compare values	k_3+ k_4 OR k_5 where $k_5 > k_4$ $\implies k_5$
Miscellaneous instructions	k_6

The temperature cooling function follows $T_i = \frac{T_1}{\ln(i+2)}$, where i is the iteration number. The asymptotic behaviour of this function as i becomes large is shown in eqn. (4.26). We have removed the "+2" factor in the $\ln(\cdot)$ and instead, equivalently, started the sum from 2 to make the analysis clearer.

$$\sum_{i=2}^{N} \frac{1}{\ln(i)} > \sum_{i=2}^{N} \frac{1}{i} \tag{4.26}$$

The right-hand sum, which is clearly smaller than the left, is the well-known divergent harmonic series (without the first term). Since the harmonic summation is divergent, so is the left sum. The harmonic series approximates to $\ln(n) + \gamma$ (where γ here is the Euler-Mascheroni constant) [110] and so has complexity $O(\log n)$, which is also a tight bound, $\Theta(\log n)$. The sum on the left of the comparison reduces more slowly than the sum on the right, so the sum on the left is lower bounded by $O(\log n)$ but must have greater worst-case complexity. The author cannot provide an analogous approximation for the sum of $1/\ln(i)$, so we assume no tight upper bound exists.

Should we instead do the analysis by assuming we continue the algorithm only until the temperature has reduced to a value of one so that we assume the algorithm's result is correct within an accuracy of a schedule by iteration count N, this would require an iteration count, as determined in eqn. (4.25). The equation (4.25) shows that there is an interrelationship between the starting temperature and the final iteration count and the algorithm complexity is decided by which of these is fixed and the extent to which that relationship is kept. We have decided to test the algorithm for various values of N and T_1 combinations, with N being the controlling variable so T_1 is a fixed value in terms of complexity and the temperature decay function does not influence the time complexity. The values were chosen so that temperature decays to a reasonably small value in the maximum iteration counts tested. The complexity is dominated per iterative step by the number of jobs n, thus the algorithm is $O(Nn)$. $\square$

4.3.4 Random Considerations

Implementation is coded in C++11 in an object-oriented paradigm using the `<random>` library. One `default_random_engine` object is used per complete run. The `<random>` class has 31 useful bits of (pseudo-)randomness per call (encoded to the integer range 1 to 2 147 483 646), which we assume do not pose a constraint. This engine is called at most twice per iteration and the number of iterations does not exceed 1 million.

It is important that the programme's internal random number generator, the random engine, be initialised carefully so that pseudo-random numbers with sufficient entropy are generated. This is the random source for the algorithm chosen so that the internal workings do not bias the result of the MCMC algorithm. The `<random>` header of C++11 provides a variety of random engine types [111] and the initialisation phase includes an instantiation of an object of one of these types. An important consideration to be made is ensuring that the random engine type chosen can in fact produce values in the required range and that the stream is long enough to produce as many values as required without recycling values. Other considerations in the choice include the performance–size trade-off. For this implementation, a Mersenne twister-type engine is selected, using the 64-bit `std::mt19937_64` source.

The obvious second important consideration is proper seeding of the engine. The header provides constructor overloads for instantiation using a default seed, or it can be passed a seed value on object instantiation. In order for the number streams to differ in every run of the MCMC algorithm, each run must have a unique seed. The library provides the `random_device` type to provide the seed. Using a `random_device` object precludes the need for seeding using time or attempting to find another source of entropy internal to the system. Since the machine on which these simulations were run has no other sound source of clearly defined and high enough entropy, we have elected to use a `random_device` object. To allow simple and rapid generation of the large number of random numbers that is required from the proposal distribution, we have built functions to perform required functions, similar to the toolkit provided in [111]. A function `std::mt19937 & global_urng()` is used to return a global instance of the Mersenne twister random engine. The function `randomize()` is created to seed the global random number generator. A generic function `getsample()` generates a proposal distribution function “on the fly” given the current sample value as the mean.

It must be ensured that the SA and MCMC random processes remain as uncorrelated as possible, and should have zero correlation in the ideal case. While the C++11 standard allows a random engine to be reused for different distribution objects and can be re-seeded, our application requires that separate engine objects are created for the MCMC process and SA runs. The same engine is used for all the runs of SA within one run of MCMC, but the SA random engine is given a different seed to start every run of SA.

4.3.5 Termination

Termination of the algorithm is determined by reaching the maximum number of iterations specified at the initialisation stage. This number was determined and verified based on the behaviour of the Geweke test results for different values, and by observation of the acceptance ratio. The results of investigation of the effect of running the algorithm for various different run lengths are given in Chapter 5. On termination, the SA algorithm outputs which section has the highest likelihood of containing an optimal solution.

The correlation of samples results in more samples being required before the set of samples has the same expectation or variance as the actual target distribution π [112]. For this reason, the term *effective sample size* has been coined defining the correction factor by which the empirical average differs from the standard variance if the samples were independent. The measure scales the sample size by the autocorrelation. We define $\tau_N = N/\rho$ as the effective sample size for the autocorrelation ρ of sequence $g(X^{(t)})$. This means that to be a sufficiently meaningful set of correlated samples, any value that would be deemed sufficient in the independent case is to be scaled in this way to get a suitable effective sample size.

The Geweke test is implemented as a test for whether the algorithm has “converged” sufficiently [46]. The means of the last 50% and first 10% of samples are compared. In general, an error percentage is defined and success is based on the error between the two sample means falling within this limit. In this work, the important work of the algorithm is to narrow down on a specific section in which SA will search. In this context, the definition of passing the Geweke test is that the two subset means indicate the same section. The acceptance ratio was also observed. Too low a ratio may indicate being very far from convergence and too high may mean insufficient exploration has taken place and being trapped in a local optimum.

4.3.6 Computational complexity of SAM

Lemma 4.2. *The MH algorithm can find a suitable section in $O(Nrn + m)$ time.*

Proof. The inputs to the algorithm are burn-in length b, the number of section divisions m, number of inner samples per section sample r, and run length N. Both m and r are variables. If the Geweke test gives a positive result, the algorithm has found a suitable section. If a suitable section can be found in N iterations after burn-in of length b for n jobs, with permutations divided into m sections, the algorithm finds a solution in

$$\begin{aligned} time \propto \{&b \times (k + r \times t(cost)) + N \times (k + r \times t(cost) + t(recording\ samples) + \\ &t(update\ proposal)) + t(bin)\} \\ &= O(bnr + Nnr + m) \\ &= O\left(Nnr + m\right) \quad . \end{aligned} \tag{4.27}$$

If, after N iterations, the Geweke test returns true and assuming the burn-in is a constant value, the complexity is $O(Nrn + m)$. □

For Theorem 4.3, we define separate iteration count variables, denoting the SA run length by N_S and the MH run length by N_M.

Theorem 4.3. *For the single machine scheduling problem, $\langle 1 \mid no\ pre\text{-}emption \mid \gamma \rangle$, a feasible suboptimal schedule can be found in no more than $O(n(N_M r + N_S) + m)$ time using the combination algorithm.*

Proof. It has been proven in Theorem 4.1 that the complexity of Simulated Annealing is $O(nN_S)$.

From Lemma 4.2, the complexity of MH is $O(N_M nr + m)$ for N_M, n, r and m all variables. Since the two algorithms are additive, the combined complexity is

$$O(N_M nr + m + N_S n) = O(n(N_M r + N_S) + m)$$

□

To perform a search through all schedules for the optimal solution requires performing the cost calculation $n!$ times and performing comparisons as many times. This has a complexity $O(n \cdot n!)$. Our algorithm provides a significant improvement to this by reducing the complexity successfully to $O(n(N_M r + N_S) + m)$. It also provides the flexibility of a number of variables that can be reduced to improve run time, or increased to improve solution quality. The solution quality–run time trade-off is dealt with in Chapter 5.

4.4 In Summary

In this chapter we have presented the formulation of the problem and detailed how it has been interpreted and moulded to facilitate the application of techniques of Metropolis-Hastings sampling and Simulated Annealing. We have provided motivation for the interpretation and selection of parameter values, and presented details about the novel SAM hybrid algorithm, including the steps involved, and implementation details. We have shown that a brute force complete search of the search space for the optimal solution is significantly more complex than the alternative algorithm we provide and have reduced the required complexity from $O(n \cdot n!)$ to $O(n(N_M r + N_S) + m)$.

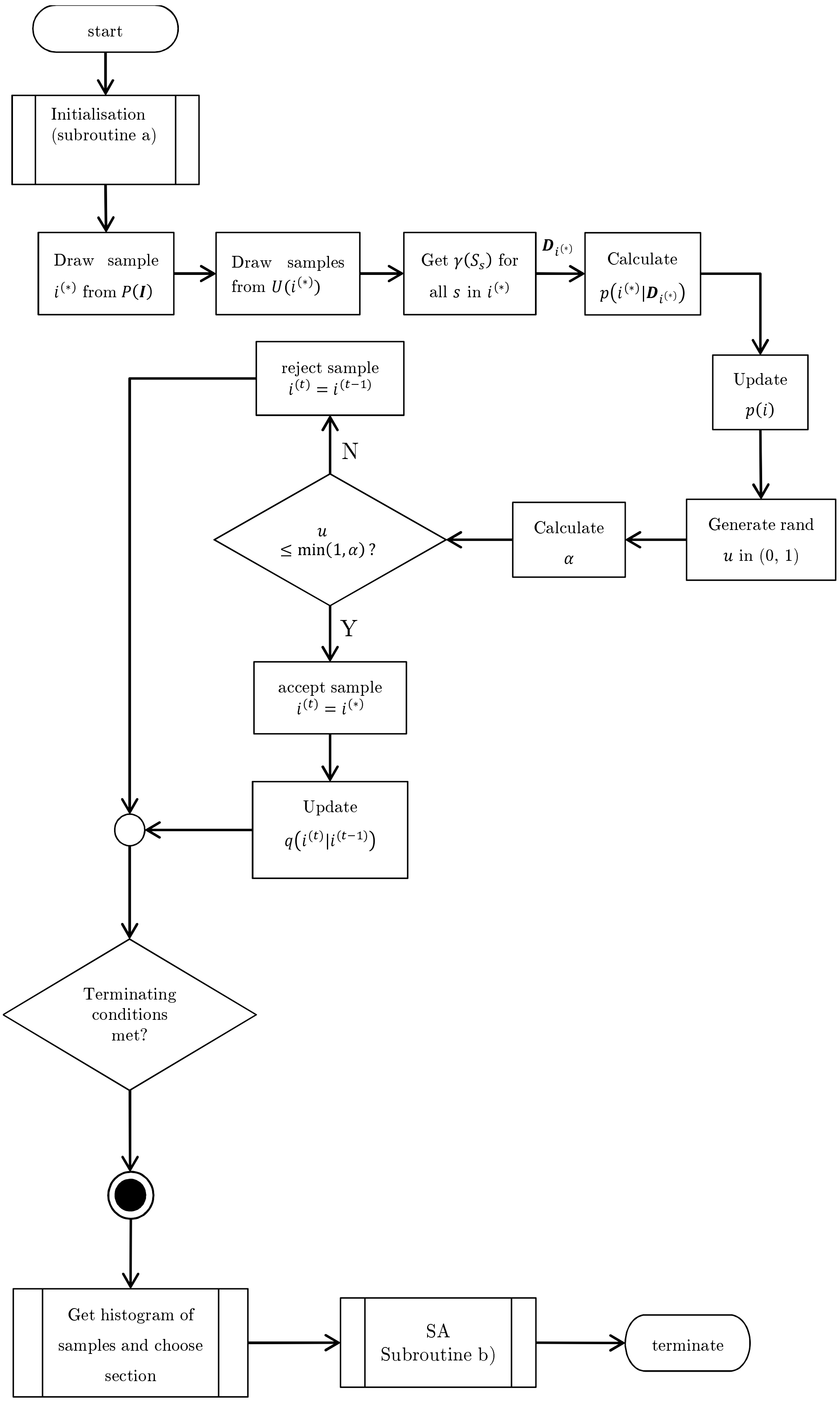

start
Initialisation (subroutine a)
Draw sample $i^{(*)}$ from $P(\boldsymbol{I})$
Draw samples from $U(i^{(*)})$
Get $\gamma(S_s)$ for all s in $i^{(*)}$
$\boldsymbol{D}_{i^{(*)}}$
Calculate $p\left(i^{(*)}|\boldsymbol{D}_{i^{(*)}}\right)$
Update $p(i)$
Generate rand u in (0, 1)
Calculate α
$u \leq \min(1, \alpha)$?
N
reject sample $i^{(t)} = i^{(t-1)}$
Y
accept sample $i^{(t)} = i^{(*)}$
Update $q\left(i^{(t)}|i^{(t-1)}\right)$
Terminating conditions met?
Get histogram of samples and choose section
SA Subroutine b)
terminate

FIGURE 4.3: Full SAM procedure

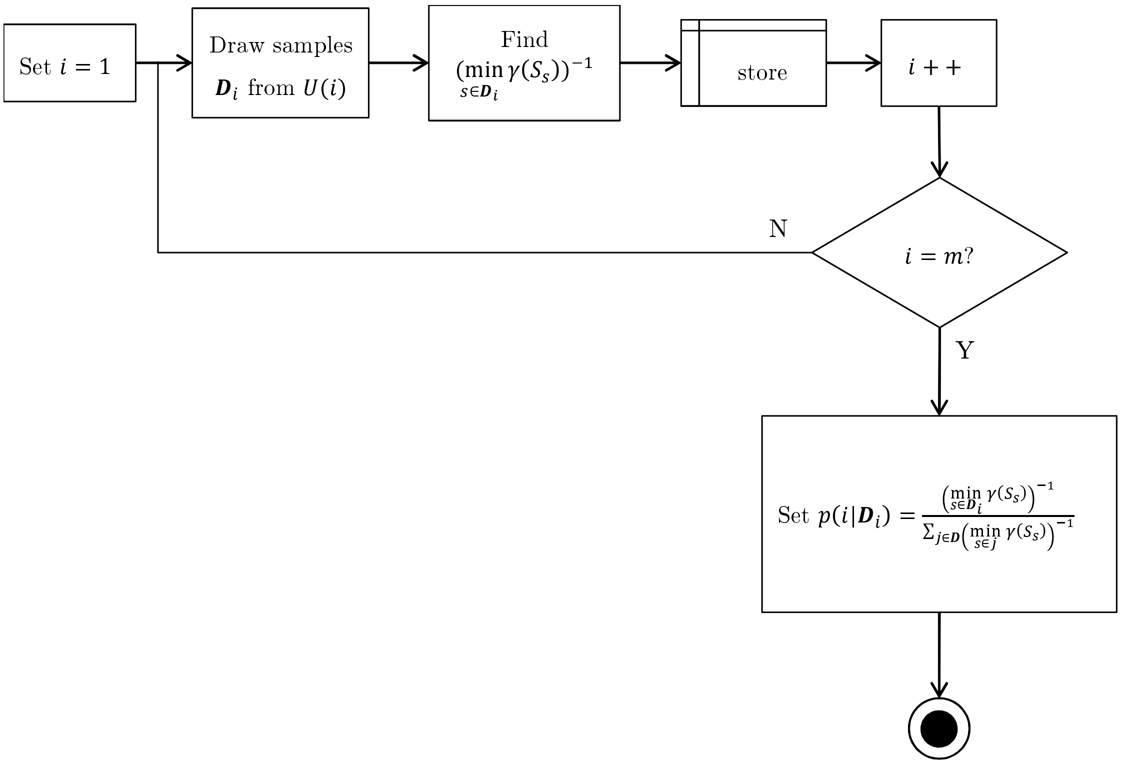

Set $i = 1$
Draw samples $\boldsymbol{D}_i$ from $U(i)$
Find $(\min_{s\in\boldsymbol{D}_i} \gamma(S_s))^{-1}$
store
$i++$
N
$i = m$?
Y
Set $p(i|\boldsymbol{D}_i) = \frac{\left(\min_{s\in\boldsymbol{D}_i} \gamma(S_s)\right)^{-1}}{\sum_{j\in\boldsymbol{D}}\left(\min_{s\in j} \gamma(S_s)\right)^{-1}}$

FIGURE 4.4: Initialisation procedure of SAM algorithm.

Chapter 5

Simulation results and discussion

5.1 Experimental methodology

5.1.1 Generating problem instances

We define a problem or problem instance by an array of processing time values, an array each of tardiness and earliness penalties (weights), and arrays for the due window start times and end times, all indexed by job number. A set of problems then consists of a number of problem instances.

We have elected to generate problems using a variant of the generally accepted method first proposed by Potts and Van Wassenhove [58] but with due windows (as opposed to due dates) as in [27, 30]. The Potts and Van Wassenhove method is widely used and accepted as a standard technique [30, 80, 82, 113–116]. While some researchers in the literature have used problem instances from open libraries (such as the OR-library [117]), these libraries either do not have the specific type of problem investigated in this work or are outdated and no longer available. Specifically, problems with due windows are difficult to find, and we have not been able to locate results of algorithms developed by other researchers that can be considered comparable to ours and tested on the specific instances for the same objective we consider. Noting that the research question requires a comparison of the performance of our implementation of the basic SA algorithm with our modified algorithms, using instances of other researchers is also not necessary to provide an adequate answer to the research question. For these reasons, it is most practical and instructive to generate our own problems using the method outlined here.

Processing times are generated from a uniform distribution in the interval [1; 100] i.e. the set of processing times $p \in [1, 100]^n$. The sizes of the n due windows are selected randomly from $U[1, \frac{C_{total}}{n}]$.

Then, defining $C_{total} = \sum_{i=1}^{p} p_i$, tardiness factor T is selected per complete problem instance from $\{0.2,\ 0.3,\ 0.4,\ \ldots,\ 0.9,\ 1.0\}$ and the Relative Due Date (RDD) is selected per problem instance from the set $\{0.4,\ 0.6,\ 0.8,\ 1.0,\ 1.2\}$. The centres of the due windows are generated from a uniform distribution with limits as indicated in eqn. (5.1):

$$U\left[(1 - T - \frac{RDD}{2}) \times C_{total},\ (1 - T + \frac{RDD}{2}) \times C_{total}\right] \quad . \tag{5.1}$$

Tardiness penalties w'' are drawn from $U[1, 10]$.

Earliness penalties w' from $U[k \times w'']$ for k in $(0, 1)$.

The full set of problem instances generated for the experiments to follow is presented in Appendix A.

5.1.2 Experimental set-up

All results must be compared against a full search of the solution space. Any heuristic is valuable mainly in its ability to yield reasonably accurate results in a fraction of the time of the incumbent method. For a full search, the cost is calculated for every possible permutation of the original schedule, that is $n!$ times, so the complexity is $O(n \times n!)$.

The performance measure is to determine the percentage deviation of the cost of the schedule produced by the tested algorithm compared with the optimum value. The quantity evaluated is percentage deviation,

$$percentage\ deviation = \frac{x - x^*}{x^*} \times 100\% \quad , \tag{5.2}$$

where x^* is the cost of the optimum schedule and x is the cost of the schedule output by the algorithm. This value is obtained for the best, average and worst (maximum) solutions in 20 to 50 runs of the algorithm. All numerical solutions are rounded to the nearest two decimal places. The shorter term *deviation* is used at times to refer to the quantity of eqn. (5.2). Where statistical standard deviation of test results is intended, the term *standard deviation* will be stated explicitly.

5.2 Simulated Annealing as the base case

We examine the statistical properties of the experimental results of the basic SA algorithm to determine the relationships between the variables and as a baseline point of comparison with the SAM algorithm we have developed. The results were obtained from 20 to 50 runs per unique combination of parameters on each of thirteen problem instances. The problem instances were generated as described in Section 5.1.2 and the variable parameter values used for the experiments are listed in Table 5.1. The "number of iterations" parameter relates directly to the algorithm run time, being the maximum iteration count for which the algorithm was run before the result was recorded.

TABLE 5.1: Parameter values used for evaluation of SA Full search

Independent variables		**Test values**
Variable name	***Symbol***	
Starting temperature	T_0	$\{0.5; 0.25\} \times n!$
Number of iterations	N	$\{1; 2; 5; 10; 50; 100; 150; 200; 500; 1000\} \times 10^3$
Start point	st	Fixed at $\frac{n!}{2}$=1814400

TABLE 5.2: Summary of averages of all results - basic SA

Temp	**907200**				**1814400**				**t-test**
N	***Max %***	***Ave %***	***Min %***	***Var***	***Max %***	***Ave %***	***Min %***	***Var***	$\|t\| > t_{crit}$
5k	7.28	3.57	0.88	13.53	7.04	3.51	0.96	12.01	F
10k	5.98	2.52	0.40	9.77	5.44	2.57	0.40	7.87	F
50k	2.67	1.23	0.19	2.00	2.55	1.16	0.11	2.00	F
100k	1.93	0.83	0.02	1.14	1.80	0.82	0.08	0.96	F
150k	1.46	0.64	0.01	0.64	1.46	0.60	0.03	0.70	F
200k	1.24	0.51	0.06	0.48	1.21	0.49	0.01	0.43	F
500k	0.79	0.25	0.00	0.16	0.48	0.14	0.00	0.26	F
1M	0.56	0.16	0.00	0.16	0.48	0.14	0.00	0.12	F

Running experiments on all 13 datasets and all variable value combinations, each for 50 runs, results in 13 000 full runs of SA. We assert that 50 is a sufficiently large number of runs to get an acceptably accurate reflection of the algorithm's performance with different combinations of parameter settings. The minimum, maximum and average percentage deviation from the optimum of the cost of the final selected schedule were recorded for the 50 runs of each problem instance, and then averaged over the 13 problem instances. The variance was also computed, which shows how longer runs (greater iteration counts) tend to converge to more similar results than shorter runs do, when comparing multiple runs with the same algorithm settings. The summary of these results are presented in Table 5.2. The right-most column is discussed in Section 5.2.1. The summary of results per problem dataset is in Section C.1 of Appendix C.

5.2.1 Significance of starting temperature on percentage deviation

In order to determine whether the starting temperature has a statistically significant impact on the percentage deviation, the t-test for two samples with unequal variances (Welch's test) was performed on the results obtained by running the algorithm on the same dataset with the same starting point and for the same iteration count, but at two different temperatures. The null hypothesis for this test is H_0: there is no significant difference in the results for the two starting temperatures, and starting temperature has no significant impact on the performance of the SA algorithm. The assumptions of the test are that the sample results are approximately normally distributed and independent. Since each run is seeded with a different pseudo-randomly generated value, the independence assumption has a high confidence. While the sample resultant costs are not strictly normally distributed (since the range is limited), we still contend that the t-test calculation can provide some elucidation into the temperature dependence. By the t-test, we reject H_0 if and only if condition

$$|t_{stat}| > t_{crit} \tag{5.3}$$

is met.

These results are presented in the right-most column of Table 5.2 above, and for all datasets in Section C.1. T indicates if condition (eqn. (5.3)) above is true and statistical significance can be claimed, and F if false. Out of 130 sets of runs comparing starting temperatures of 907 200 (a quarter the size of the search space) and 1 814 400 (half of the size of the search space) only 7 instances returned a true t-test result i.e. have a statistically significant difference. For the other 123 sets of results, $t_{stat} \not< -t_{crit}$ and $t_{stat} \not> t_{crit}$. Three of the instances where statistically significant differences can be observed are in the lowest iteration count ($N = 1000$ iterations), as can be expected, since in shorter runs the range of exploration that the algorithm is able to do is much smaller than in longer runs. The higher starting temperature appears to be superior in the case of very short runs but the data is insufficient to make a conclusive claim. The other instances do not show any significant pattern. There is not enough evidence to reject the null hypothesis and we conclude that starting temperature does not affect the results significantly for the iteration counts and temperatures tested. This conclusion suggests that comparisons of basic SA with our hybrid algorithm SAM in the following subsection will not be skewed by a dependence on the starting temperature.

5.2.2 The effect of iteration count on percentage deviation

By glancing through the results in Table 5.2, we can observe that the accuracy of results of SA improves considerably and steadily from runs of 1000 iterations to longer runs of 1 million iterations. As a precaution, however, we first perform hypothesis testing to determine whether the maximum iteration count has an effect on the percentage deviation of the final results. To determine this, a (multi-column or array) Chi-squared test has been done on each dataset, comparing the results with those that would have been obtained if selections followed pure chance. The p-values are presented below in Table 5.3. Since for all values, $p \ll 0.001$, we can conclude that iteration count does indeed have a statistically significant impact on the percentage deviation obtained.

Table 5.3: Significance of iteration count by p-values per dataset

Dataset	***p*-value**
0	4.42E-26
1	2.2E-109
2	5.3E-101
3	6.6E-124
4	7.7E-105
5	1.49E-53
6	7E-47
7	1.36E-28
8	9.59E-31
9	6.43E-30
10	7.19E-43
11	2.34E-48
12	6.22E-41

Table 5.4 presents the 99th percentile of the maximums of the results of all datasets to show what expected accuracy can be anticipated by choosing a suitable number of iterations for which to run the algorithm. Notably, the results show that a high iteration count is required to ensure that results are close to the optimum solution. Specifically, the 99th percentile of the maximum deviation values reveals that if an application requires that the result be no more than 2% deviant from the optimum with a 99% likelihood, this would generally require runs of length between 150 000 and 200 000 iterations. If a 5% or less deviation is required, this would probably require around 40 000 iterations. These results were obtained by calculating the 0.99 percentile of all the worst-case (maximum) deviations (in %) for all runs of all problem sets, for the solutions produced by the algorithm run for various iteration counts. The goal

Table 5.4: 0.99 percentiles of maximum percentage deviations for run lengths given by the iteration count

Iteration count	**0.99 percentile**
1k	16.68
2k	13.47
5k	8.53
10k	6.82
50k	3.24
100k	2.13
150k	2.05
200k	1.52
500k	1.03
1M	0.61

of the SAM algorithm we developed and present in the next few sections is to produce results with similar accuracy in significantly fewer iterations (shorter runs), to reduce the running time and effective complexity. This is because the time sensitivity in applications such as communications makes algorithms with longer run times impractical.

5.3 SAM

5.3.1 Coarse Metropolis-Hastings

As per Chapter 4, the aim of the SAM algorithm is to add in a step of pre-sampling by MH before running SA on the pruned search space to reduce the overall running time required to get a result within 5% of the optimum. The outline of coarse MH sampling is to divide the total search space of possible schedule permutations into a number of sections and use MH sampling to find a piecewise-defined distribution over the sections, and then to determine the most likely section to contain an optimum schedule. There are three main parameters we examine in the MH implementation: number of sections (m), the algorithm run length or maximum number of iterations (N), and the number of inner samples taken for every section sampled in MH (r).

The set of number of sections used in the experiments is

$$\mathbf{m} = \{20,\ 50,\ 100\} \quad . \tag{5.4}$$

The algorithm was run using each of the values in this set, for the run lengths of iteration counts in the set

$$\mathbf{N} = \{500,\ 1\,000,\ 2\,000,\ 5\,000,\ 10\,000,\ 50\,000\} \quad . \tag{5.5}$$

The third variable to investigate was the number of uniform inner samples taken per section sampled. The number of samples investigated are in the set

$$\mathbf{r} = \{50,\ 100,\ 200,\ 500\} \quad . \tag{5.6}$$

The important interactions we explore are the number of iterations vs. number of sections (for the same number of inner samples), the number of iterations vs. number of inner samples (for constant number of sections), and number of sections vs. number of inner samples for the same iteration count.

Before embarking on these investigations, however, the Chi-squared test was performed on every set of MH section results to confirm that the method does, in fact, provide useful information that has a statistically significant difference from if it were constructed by pure chance (the case of maximum entropy).

5.3.1.1 Different runs with the same settings

Different runs of the same algorithm with exactly the same parameter settings produce different results each time and so the algorithm was run 20 times on each unique combination of parameters. For our purposes, it is desirable that the differences between runs be small enough so that the same section is most likely to be selected in each run. If the spread and, thus, variance of the section chosen as most likely is large, the algorithm does not prove as useful, since it does not supply us with valuable information. In such a case, the selected section result cannot be trusted and one may be better off running SA on the full search area for a large number of iterations.

Figures 5.1 and 5.2 show the normalised frequency distribution histograms, each of five different runs – represented by the different coloured bars – of the MH algorithm with the same parameter settings, for two different problem instances. For this example, the search space is divided into 10 sections, the iteration count is 500 (excluding burn-in), and the start point is at the midpoint of the search space. In Figure 5.1, it can be seen that the frequency distribution is different for the five runs but that the section with the highest frequency is still fairly clear and similar for all runs. Figure 5.2 shows five runs for a different problem set with equal parameter settings to those of Figure 5.1. The results for this problem instance are more diverse such that runs produce different

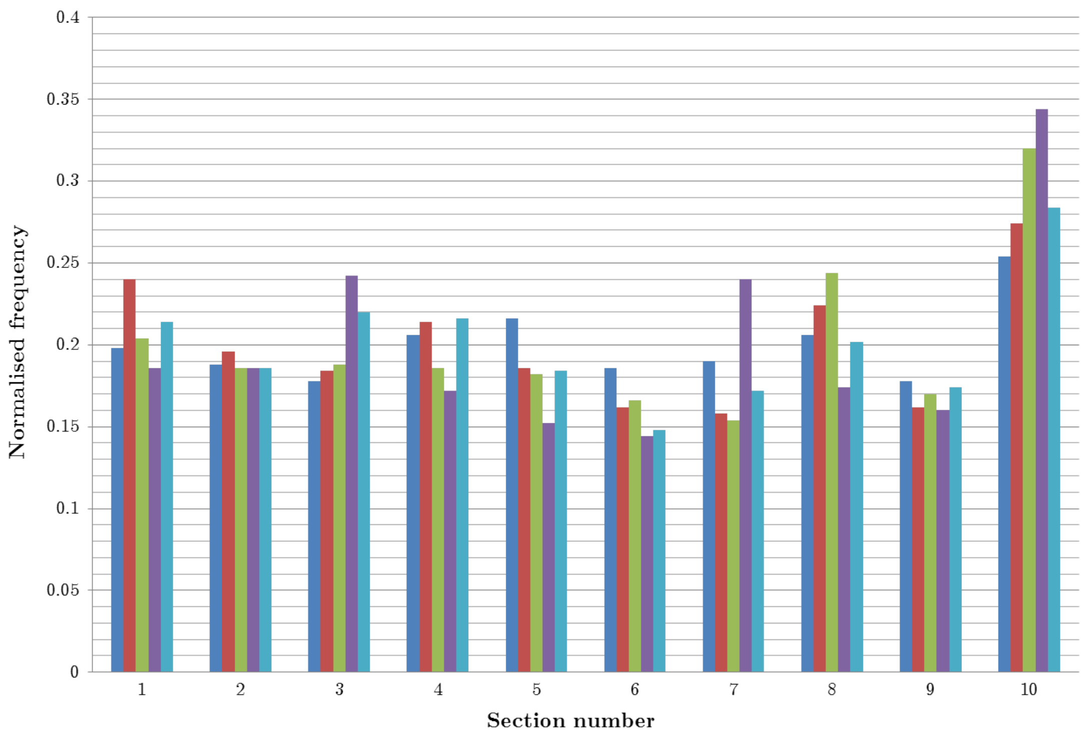


FIGURE 5.1: Normalised frequency distribution by Metropolis-Hastings sampling: problem set with smaller variance

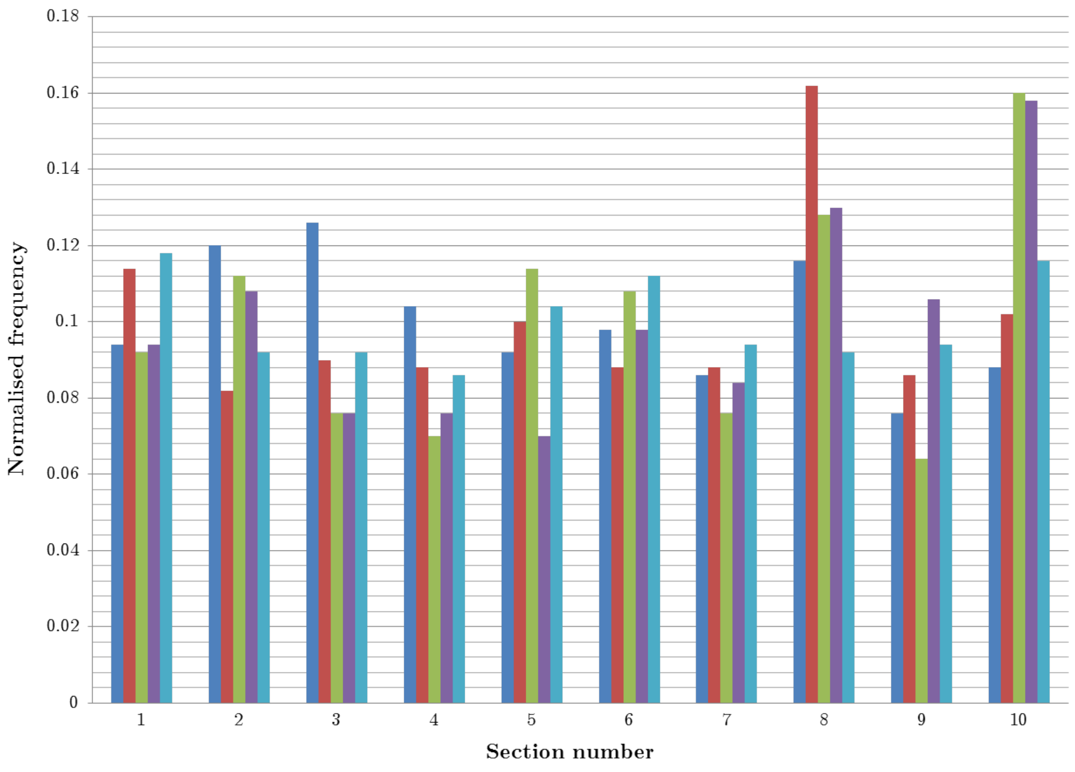


FIGURE 5.2: Normalised frequency distribution by Metropolis-Hastings sampling: problem set with larger variance

selected sections. In this figure, the variance of the chosen section is large. This may have a negative impact on the overall SAM result. This is the danger of running the algorithm for too short a duration, and illustrates that performance may be notably different on different datasets. More examples are in C.2.2. Table 5.5 also shows the number of times out of 20 runs that each of 20 sections was chosen, illustrating the varying results per run. Whether the larger variance significantly affects the final result of SAM is explored in Section 5.3.2.

Table 5.5: Unnormalised frequency with which each section (out of 20 sections) was selected in 20 runs of MH, with the p-values of Chi-squared test: Dataset 0

N		**500**	**1000**	**2000**	**5000**	**10000**	**50000**
section	H_0	H_a	H_a	H_a	H_a	H_a	H_a
1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0
5	1	0	0	0	0	0	0
6	1	0	0	0	0	0	0
7	1	0	0	0	0	0	0
8	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0
10	1	0	0	0	0	0	0
11	1	0	0	0	0	0	0
12	1	0	0	0	0	0	0
13	1	0	0	0	0	0	0
14	1	0	0	0	0	0	0
15	1	0	0	0	0	0	0
16	1	0	0	0	0	0	0
17	1	0	0	0	0	0	0
18	1	4	4	6	2	4	5
19	1	16	16	14	18	16	15
20	1	0	0	0	0	0	0
***p*-value**		1.2E-42	1.2E-42	1.4E-34	4.6E-54	1.2E-42	3.4E-38

The Chi-squared test was performed on every set of MH section results to determine that the method produces results with a statistically significant difference compared to the situation if sections were selected by pure chance (the case of maximum entropy). An example is given in Table 5.5 to illustrate the methodology. The table shows the number of full runs of MH out of 20 in which each section number was selected as the most likely candidate to contain an optimum schedule (taking $r = 50$ inner samples per section sample in this example). The benchmark against which these relative frequencies were tested, representing the null hypothesis (H_0) that there is no difference in the frequencies with which sections are chosen, is pure chance where each section is equally likely to be chosen. For 100 sections, therefore, H_0 is 0.2 out of 20 runs for all sections. In the case of 50 sections, it is 0.4 out of 20 runs and for 20 sections it is 1 out of 20. The frequencies for each combination of parameters were compared against H_0 by running the Chi-squared test.

Since the Chi-squared test proved $p \ll 0.0001$ for every set of runs for every dataset and all combinations of parameters (see Appendix C.2.1 for all), we can conclude, at the very least, that MH provides a large information content[1] that may be helpful in determining where the optimum, or a good schedule, is least and most likely to be found.

5.3.1.2 Impact of iteration count

We now present results of the frequency distributions of selected sections, illustrating the impact of iteration count on the section selected in the MH step. To illustrate, histograms are presented in Figure 5.3 showing the number of times out of 20 runs or the unnormalised frequency with which each section was selected. The figures are all of results where the maximum number of inner samples (500 per section samples) was used to control the variable and with 20 sections, simply because 20 sections is visually clearer than cases of more sections. Only a sample of the results is shown here, i.e. dataset 0 (Figure 5.3a), dataset 5 (Figure 5.3b), dataset 8 (Figure 5.3c) and dataset 10 (Figure 5.3d). The full set can be found in Appendix C.2.2.

There is a strong observable correlation between the results of longer and shorter runs, with the same two or three sections selected every time, whether MH was run for 1000 or 50 000 iterations. We may observe a slightly lower variance in the chosen sections of longer runs than shorter but these differences were tested not to be statistically significant nor do they fulfil the requirement of repeatability since the effect is not present in the results of all datasets. This means that we can get away with shorter runs of 1000 iterations without a drop in solution quality – a positive indicator to the usefulness of this kind of pre-sampling because it does not add significant computation to the algorithm.

5.3.1.3 Impact of number of samples

Figures 5.4 to 5.7 are histograms of the final most likely section chosen in 20 different runs of MH per dataset, for increasing numbers of uniform inner samples in **r** per section sample of MH, in runs of different lengths. The x-axis shows the sections: either 1 to 20 for examples of 20 section divisions or 1 to 50 for 50 sections. The y-axis shows the number of times out of 20 runs that the corresponding section was chosen, representing an unnormalised frequency. The different coloured bars are for runs of different lengths in the set N, defined by eqn. (5.5).

The figures illustrate a trend that the sections chosen from the distributions generated by MH become more concentrated as the number of inner samples per MH section sample increases. Looking at Figure 5.4a of dataset 1 for 50 inner samples, the sections chosen are diverse with selections of section 3, a few between sections 26 and 29, 36 and 39 and 46 to 49. Figure 5.4b with 100 inner samples is more concentrated on sections 46 to 50 with no selections in the range 36 to 39. Figures 5.4c and 5.4d (200 and 500 inner samples, respectively) are yet more concentrated with all selections being in the range 46 to 50 by the latter. The same trend is clear from the figures of dataset 5 (Figure 5.5). The narrowing in as the number of inner samples increases is even more pronounced in dataset 10 (Figure 5.6) as Figure 5.6a shows a large assortment of chosen sections, all with significant frequencies while Figure 5.6d shows much less varied resultant section selection. The more inner samples are gathered, the more information is provided to the algorithm about the properties of the section from which these samples are taken, and the clearer its understanding of the cost distribution, and thus, the likelihood of optimal or near-optimal solutions being contained in each section. There is a trade-off to be made between increased inner samples providing greater clarity on the section to use for SA and the increased computational load introduced by the requirement to take and store more samples, which compounds as these are taken at every iteration of the outer MH loop.

Looking at Figures 5.4 to 5.7 and comparing them to Figure 5.3, it appears that the number of inner samples has a more profound effect than the iteration count on the chosen section, at least for the values used in these experiments. As the inner sample count increases so the range of chosen most likely section reduces and narrows down on fewer options. This is because the more samples that are gathered, the more accurate the inference is about the value representing the likelihood value of the sample's section is. The effect is especially pronounced in Dataset 10, although it is also visible in all other sets. It is curious that there is very little significant difference in the results

[1]This refers to the formal definition of information content as defined in terms of entropy by Shannon, “A Mathematical theory of communication” [118]

when increasing iteration count and suggests, surprisingly, that better results could be obtained by doubling the number of inner samples than doubling the iteration count – both of which have the same effect on run time (see Lemma 4.2).

5.3.1.4 Impact of number of sections

Considering the number of sections variable, with constant number of inner samples and all run for lengths of 50 000 iterations, a general correlation between spread of chosen section and number of section divisions is observable. This uncertainty makes division into greater numbers of sections undesirable. The smaller the section size (more sections) used in the MH step, the easier it is for SA to find the optimal in a short time in the second SA step. On the other hand, the larger the number of sections, the greater the error, requiring longer runs of MH to reduce this error. Only using 10 sections produces the same selected optimal section for all runs of the algorithm on the same dataset. Indeed, the correct section was selected in as few as 500 iterations for the majority of the problem sets tested with 10 sections.

This result can be used to find trade-offs between smaller sections resulting in less uncertainty and variability in the outcomes of SA reduced search but with reduced exploration opportunity resulting in increased probability of error; and larger sections enabling more exploration by reduced SA but greater variability. A suitable trade-off to ensure low errors while still providing a significant decrease in effective complexity on the final SAM results is discussed in the next subsection.

5.3.2 Reduced search Simulated Annealing

Table 5.6 presents the summarised results obtained when running the SA algorithm on the reduced search space selected by the coarse MH algorithm for cases of 20, 50 and 100 sections, averaged over all runs for all problem sets. A complete set of results alongside those of the basic "full search" SA algorithm can be found in Table C.7 of Appendix C.3. In the tables, SAM20 refers to SA reduced search with 20 section divisions of the search space, SAM50 refers to 50 sections and SAM100 to 100 sections. Results with the label T1 are those of basic SA using a starting temperature of 907 200 (i.e. a 1/4 of the size of the search space), and T2 refer to results with starting temperature of 1 814 400 (1/2 the size of the search space).

The SA algorithm was run 30 times on each of the sections that were selected in the previous MH step on the thirteen data sets, for each of the run lengths shown by **N**. The temperature for reduced search space SA was set to exactly half the size of the

Table 5.6: Average percentage deviation of reduced search SA results compared with full search basic SA

	Basic SA T1		**Basic SA T2**		**SAM20**		**SAM50**		**SAM100**	
N	***Avg %***	***Var***	***Avg %***	***Var***	***Avg %***	***Var***	***Avg %***	***Var***	***Avg %***	***Var***
1k	7.84	168.78	6.98	47.72	6.19	19.74	5.82	29.03	1.83	9.89
2k	5.31	39.80	5.06	33.42	5.48	16.94	5.20	17.43	1.34	5.60
5k	3.57	13.53	3.51	12.01	4.82	8.33	4.59	12.07	0.91	2.87
10k	2.52	9.77	2.57	7.87	4.48	6.24	4.36	11.13	0.71	1.92
50k	1.23	2.00	1.16	2.00	4.03	4.21	4.06	10.12	0.52	1.25
100k	0.83	1.14	0.82	0.96	3.94	3.78	4.01	9.84	0.50	1.12
150k	0.64	0.64	0.60	0.70	3.91	3.70	4.00	9.88	0.50	1.10
200k	0.51	0.48	0.49	0.43	3.89	3.65	3.99	9.81	0.50	1.08
500k	0.25	0.21	0.25	0.26	3.86	3.46	3.99	9.80	0.50	1.08
1M	0.16	0.16	0.14	0.12	3.85	3.39	3.99	9.80	0.50	1.08

sections, and the start point was set to the midpoint of the section. The maximum (max), minimum (min) and average (avg) deviations of the selected schedule calculated over the set of 30 runs were recorded for all section sizes and all datasets. Averages were calculated as weighted averages taken over all the runs performed on that section division number (20, 50 or 100) weighted according to the frequency with which the section was selected by MH sampling as having the highest likelihood. For example, for dataset 0, the summary of the results of coarse MH sampling for 20 sections is given in Table 5.5 in Section 5.3.1. This shows how many times each section number from 1 to 20 was chosen out of the 20 runs performed for run lengths of the iteration count indicated. For 2000, 5000 and 50 000 iterations, the result was section 18, 30% of the runs and section 19, 70% of the runs; section 18, 10% of runs and section 19, 90% of runs; and section 18, 25% of runs and section 19 75% respectively. These percentages are the weights by which the average percentage deviation values obtained by the SA search in those respective sections are multiplied in the weighted averaging calculation of Table 5.6. In other words, the deviations per run were averaged over the 30 runs, and multiplied by the section weight. The variances were calculated directly from the data. The result of performing Welch's test for significance comparing the % deviation results of basic SA with those of SAM are in the columns headed $|t| > t_{critical}$ in the Table C.7 in Appendix C.3. This column shows whether the differences in the results of basic SA compared with those of the two-step SAM algorithm are statistically significant.

For ease of visualisation of the relationships, the mean values obtained from a basic SA with starting temperature of 1/4 of the search space (907 200), basic SA with starting temperature of 1/2 the search space, SAM with 20 sections, 50 sections and 100 sections are plotted together in Figure 5.8 along with the averages of all the datasets' maximum values. These are denoted by Fullsearch 0.25N, Fullsearch 0.5N, SAM20, SAM50 and SAM100 respectively where N *here* (and only here) is the total number of points in the search space ($n!$). For the results of all datasets, please refer to Figures C.4 to C.16.

5.4 Discussion

5.4.1 Overview of Results

From the results, it is clear that for shorter runs our modified SAM algorithm yields significantly better results than basic SA. The point at which basic SA starts to catch up in longer runs differs per dataset from 10 000 to 200 000 iterations in length. If this algorithm is to be deployed in network managers for scheduling packets in wireless networks, the time cost of runs longer than 10 000 iterations is not acceptable to provide QoS as network delays will be too long. In this situation our SAM algorithm is clearly the preferred choice, as it can yield deviations as low as 0.15% in a mere 1000 iterations.

Figures 5.8 and C.4 to C.16 illustrate that SAM is superior to basic SA when considering lower iteration counts and converges on lower percentage deviations in the same number of iterations, whether SAM20, 50 or 100 sections. In particular, SAM100 generally produces the lowest percentage deviation results for shorter runs, up to 5k (see Figures C.7, C.8, C.12) and even up to 10k iterations (Figures C.5, C.9, C.16), 50k iterations (Figures C.4, C.6, C.10, C.11) and 200k iterations (Figure C.14). The worst-case values of both basic SA algorithms are significantly higher than the SAM results for run lengths less than 5000 iterations, emphasised by the pre-eminence of True Welch test outcomes for the results showing statistical significance of the difference between basic SA and the superior SAM algorithm. SAM20 still performs well even when compared to long runs of basic SA up to 1 million iterations in length. On dataset 7, SAM performs better than basic SA throughout, with SAM20 performing significantly better even when compared to basic SA runs of 1 million iterations.

There is a point in run lengths where the two methods cross over and it is more effective to run basic SA longer than to use SAM. We also see from Table 5.6 that the differences in % deviation results are likely to be significantly better for SAM than basic SA on the lower iteration count spectrum and significantly better for basic SA than SAM at the high iteration counts but in the middle iteration count ranges, the performance is not generally statistically significantly different. This point is where the difference becomes statistically insignificant and the t-test returns false (shown in Table C.7) and is generally between 100 000 and 150 000 iterations. However, as an example we note that the execution time measured to run the SA algorithms for 150 000 iterations was of the order of 5 s. The acceptable maximum delay for Voice over IP or video stated in Request For Comment (RFC) flow specification 1363, and telephony according to the ITU-T recommendation G.114, is 150 ms [119]. This means that even if our system and programming style or environment has bloated the execution time by a factor of 10, this would still cause an unacceptable delay of 500 ms in data traffic for the purpose of VoIP or video in a network. We can recommend the use of short runs of SAM instead. Still, we note that scheduling optimisation may only be performed less regularly between large batches of packet transmissions so the longer time becomes more acceptable. Also, with increasing research in high speed network cards, the complexity may become less of a problem and the time delay reduced.

If the accuracy of the MH results is good then SAM can provide significant improvements but when a bad suboptimal section is chosen such as when the algorithm does not escape from a local minimum, the deviation can be quite large (such as can be observed in dataset 11, 50 sections). Large deviations after long runs are particularly prevalent in SAM20 and SAM50, as can be seen in both Figures 5.8 and C.5, C.14 and C.15. Even in this instance, however, it must be noted that the deviation does not exceed 3.5% in the cases where the deviation is larger than that of basic SA. This performance is still quite acceptable. On the other end of the spectrum even for large sections (SAM20) very small deviations can be obtained (such as observed in Figures C.7 and C.8). For different datasets and depending on the sections chosen by MH there is no clear winner between 20, 50 or 100 sections as they each perform differently on the datasets tested.

Together then it is observed that if the coarse MH algorithm is run for 1000 iterations, reducing the search space by a factor of 20 and then running SA on the reduced search space for 10 000 iterations, a similar deviation can be achieved as running SA on the full search space for 50 000 iterations (about 5%). The improvement achieved is reducing the run time by a factor of 4.5. While this may seem modest, it must be noted that job sets to be scheduled will typically be far larger than ten as used in all our experiments. For 50 jobs, let us assume that the full search SA will require 1 million iterations. The cost calculation is an unavoidable bottleneck with a complexity $O(n)$, which would need to be calculated 1 million times. SAM could provide the solution in less than a total of 222 223 iterations including as many cost calculations or less.

5.4.2 Comparisons

Both basic SA and SAM are significant improvements from the brute force search approach, which has a complexity of $O(n \cdot n!)$. A search for a problem of 50 jobs using the brute force search requires the cost calculation to be made of the order of 10^{64} times. SA is a great improvement from this, and our SAM algorithm an even greater improvement.

The benchmarks provided by Biskup and Feldmann of average percentage deviation from the optimal of 70 different 10-job problem instances with common due dates was 2.28% [120], which our SAM100 algorithm outperforms at 1.83% when run for only 1000 iterations. This is despite the problem being more difficult, since there are more variables to consider (the different due windows).

M'Hallah's hybrid GA-Hill Climbing algorithm for minimising total single machine earliness-tardiness [121] has a complexity of

$$\begin{aligned} time &\propto P \cdot G \cdot O(fitnessfunction) \cdot (O(crossover) + O(mutation) + O(replacement)) \\ &\propto O(P \cdot G \cdot n(2s + 2 + 2n)) \\ &= O(PGn(s + n)) \quad , \end{aligned} \tag{5.7}$$

where
N is the number iterations or steps,
P is the population size,
G is the number of generations,
n is the gene size (number of jobs), and
s is the size of the subsequence chosen for cross-over.

This complexity does not show the number of constant-time operations that have a high load, causing the whole algorithm to have long running times in practice. If we consider, for each simulation of SAM, that we keep the number of inner samples and the number of sections constant, this complexity of the hybrid GA-Hill Climbing algorithm is much higher than SAM's $O(n(N_M + N_S))$. This complexity also omits that the initial population is generated by a greedy heuristic.

The datasets for testing were generated using the same framework as we did [120] so we loosely compare results for 10-job schedules. Three different variants of greedy heuristics yielded average results of 11.80%, 14.58% and 60.13% deviation from the optimal, respectively, but the number of iterations required to produce these results is not clear. This we can compare to our average results of SAM20, SAM50 and SAM100 of 3.85%, 3.99% and 0.50% respectively for a run of 1 million iterations. The rest of the results are not given in terms of deviation from the optimum and so defy further comparison.

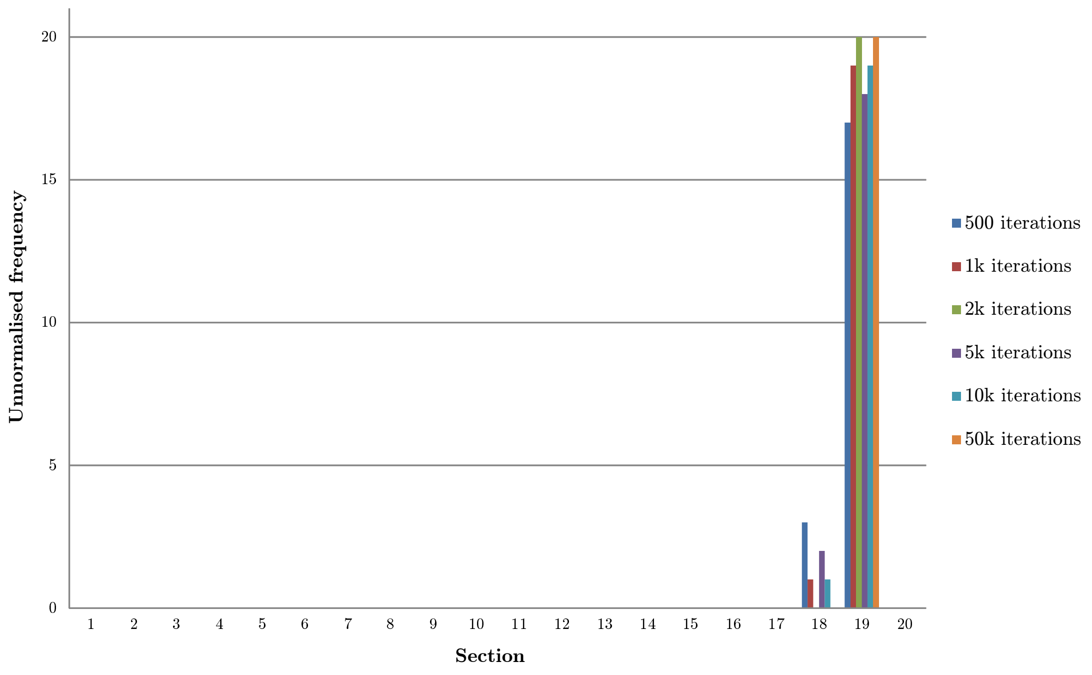
A multi-objective SA algorithm is hybridised with an evolutionary algorithm by Yannibelli and Amandi [81], but the differing objectives make comparison with our work less instructive. In particular, the fitness function depends on the optimisation objectives so the complexity of this step cannot be compared fairly. The hybrid algorithm, however, includes a full run of SA performed on every generation. This inevitably increases the complexity by the number of SA iterations, since it is multiplied by the basic evolutionary complexity instead of being added to the pre-sampling step as we do.

Hino et al. test hybrids of TS and GA on minimising earliness and tardiness penalties with a single *common* due date [33]. Numerical results of running the proposed algorithm on problems generated by the usual Biskup-Feldmann method [120] had mean percentage deviations of between 1.53% and 22.97% for 10-job problems.

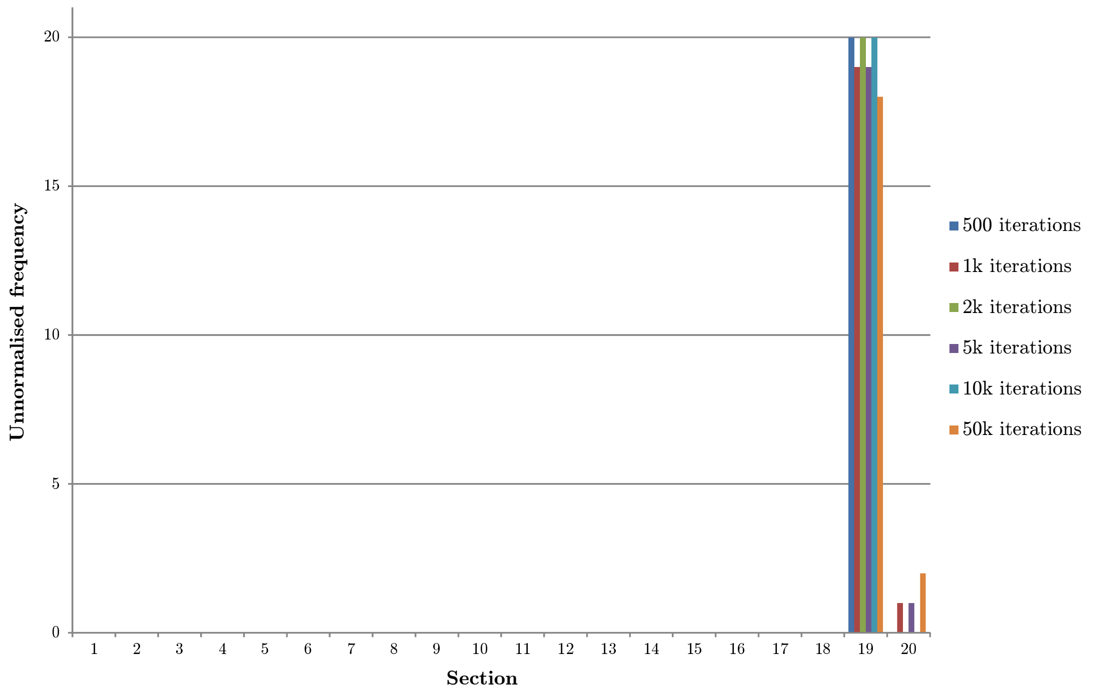
The hybrid heuristic of Ali and Bijari for minimising maximum earliness and tardiness with distinct due dates had average percentage deviations of between 0.0% and 3.34% for 10-job problems constructed by the Biskup-Feldmann method [82]. Their heuristic makes use of the Kuhn-Munkres Hungarian algorithm.

GA is alternated with EDA to improve the effective complexity of EDA, which may result in a complexity of $O(n^2)$ [80]. The algorithms are alternated in different proportions to determine an effective mix ratio. A similar method is used for problem generation but problem sizes tested are from 20 to 90 jobs with runs of 50 000 to 125 000. Results are given as absolute values instead of percentage deviation so comparison cannot be made.

Other hybrid algorithms are used for parallel machine scheduling [122, 123], flow-shop [124] and single-machine total tardiness scheduling with breakdown interruptions [125], none of which can be effectively compared with our work.

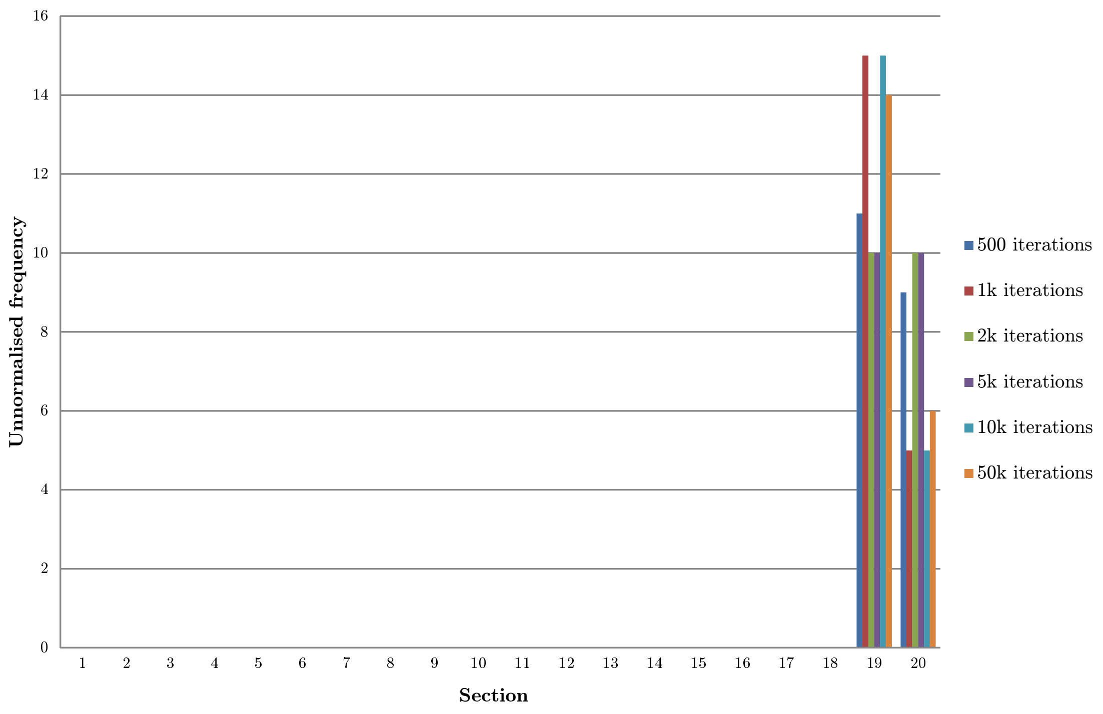


(a) Dataset 0

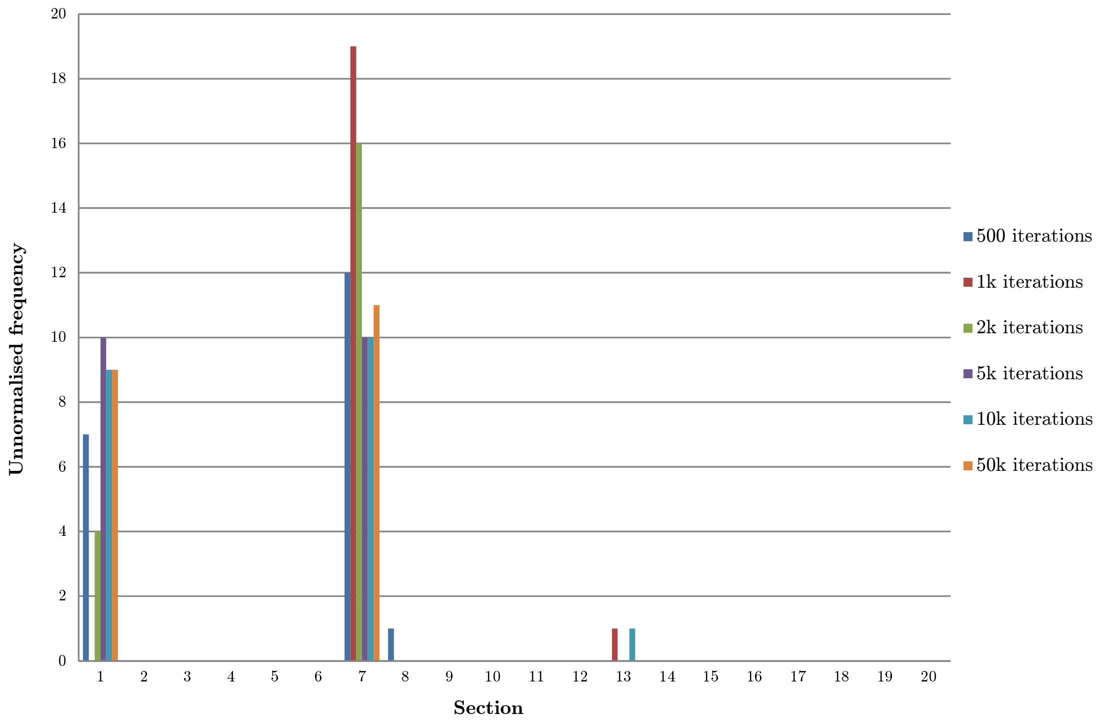


(b) Dataset 5

Figure 5.3: Histogram showing the effect of iteration count on the sections selected by MH: 500 inner samples and 20 sections

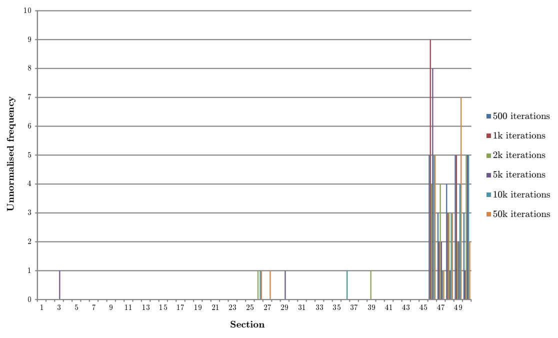


(c) Dataset 8

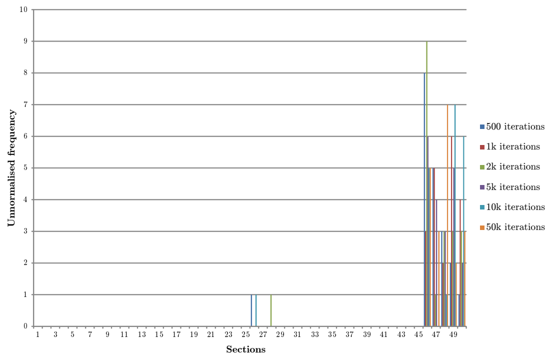


(d) Dataset 10

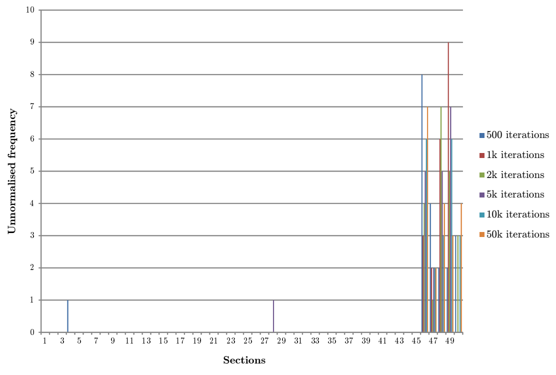
FIGURE 5.3: Histogram showing the effect of iteration count on the sections selected by MH: 500 inner samples and 20 sections



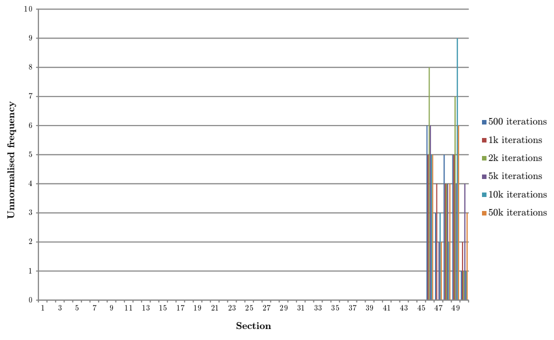
(a) Dataset 1 with 50 inner samples



(b) Dataset 1 with 100 inner samples

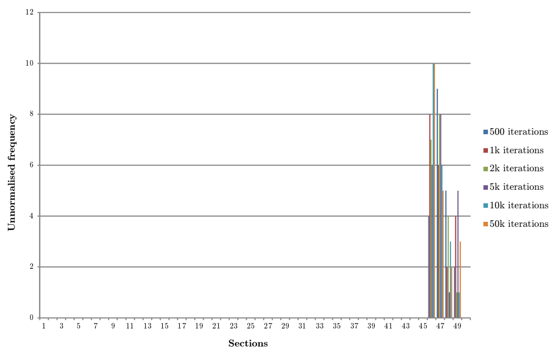


(c) Dataset 1 with 200 inner samples

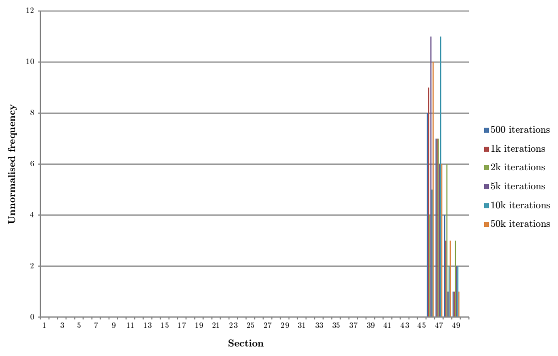


(d) Dataset 1 with 500 inner samples

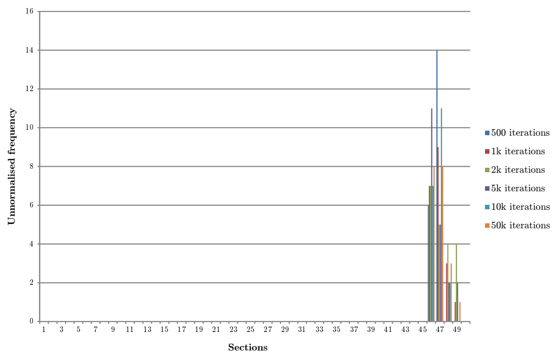
Figure 5.4: Histogram showing the distribution produced by MH with 50, 100, 200 and 500 inner samples, dataset 1. Higher numbers of inner samples result in more concentrated distributions



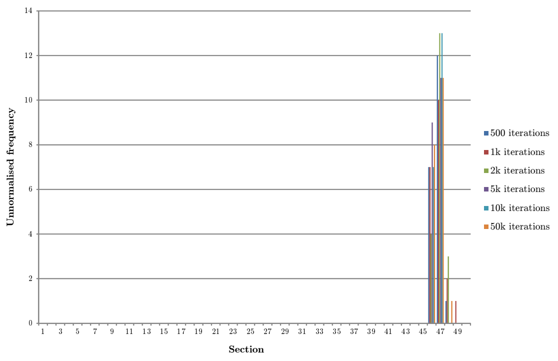
(a) Dataset 5 with 50 inner samples



(b) Dataset 5 with 100 inner samples

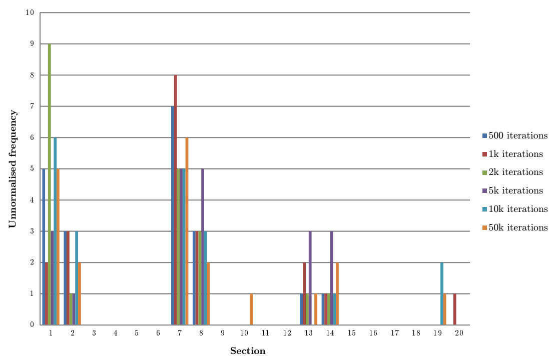


(c) Dataset 5 with 200 inner samples

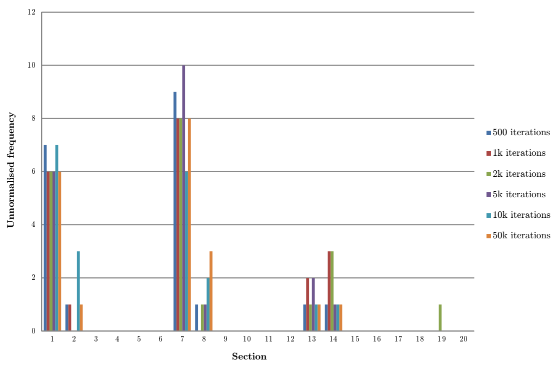


(d) Dataset 5 with 500 inner samples

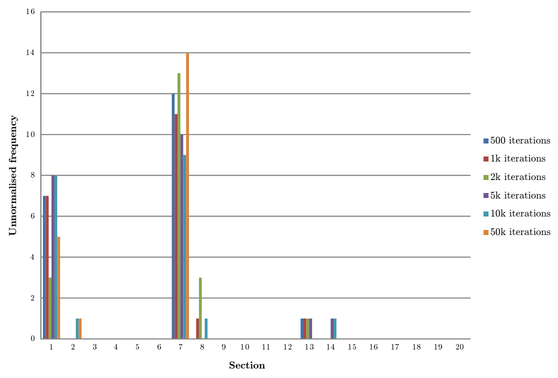
FIGURE 5.5: Histogram showing the distribution produced by MH with 50, 100, 200 and 500 inner samples, dataset 5. Higher numbers of inner samples result in more concentrated distributions



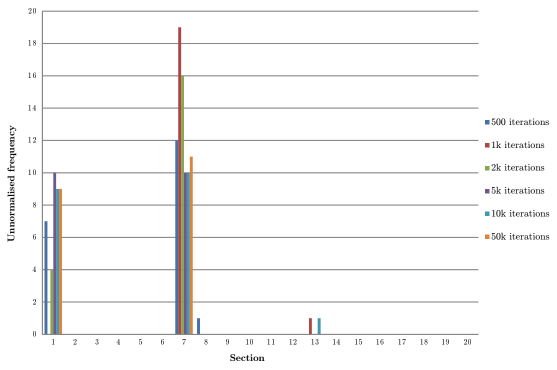
(a) Dataset 10 with 50 inner samples



(b) Dataset 10 with 100 inner samples

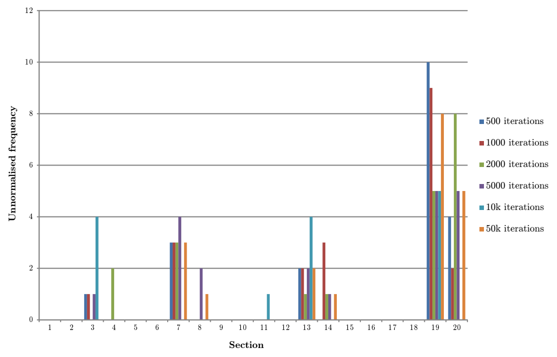


(c) Dataset 10 with 200 inner samples

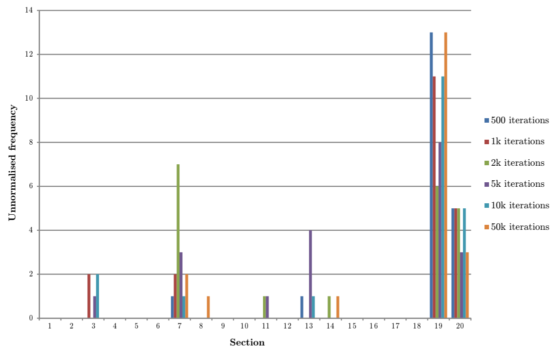


(d) Dataset 10 with 500 inner samples

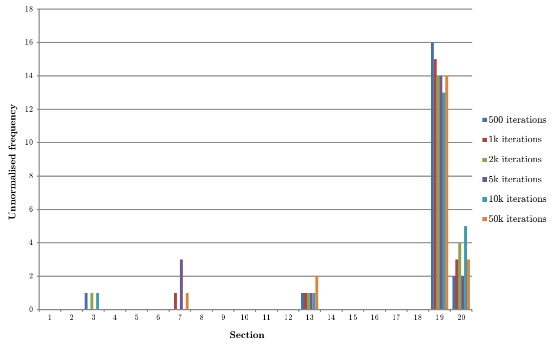
FIGURE 5.6: Histogram showing the distribution produced by MH with 50, 100, 200 and 500 inner samples, dataset 10. Higher numbers of inner samples result in more concentrated distributions



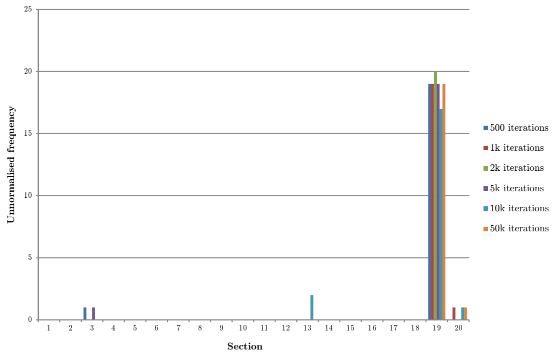
(a) Dataset 13 with 50 inner samples



(b) Dataset 13 with 100 inner samples

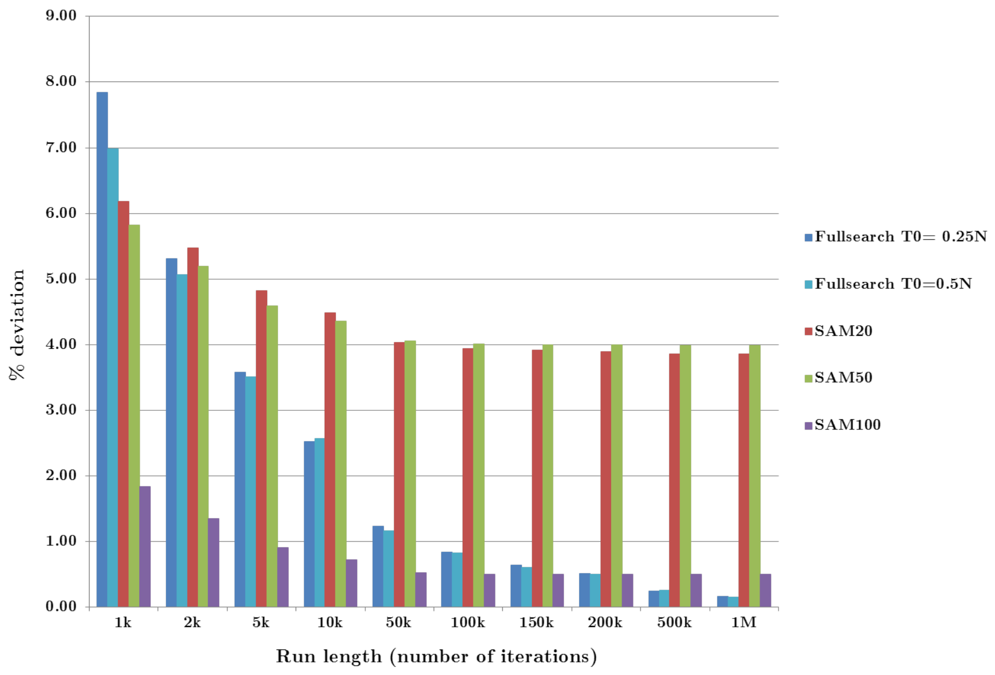


(c) Dataset 13 with 200 inner samples

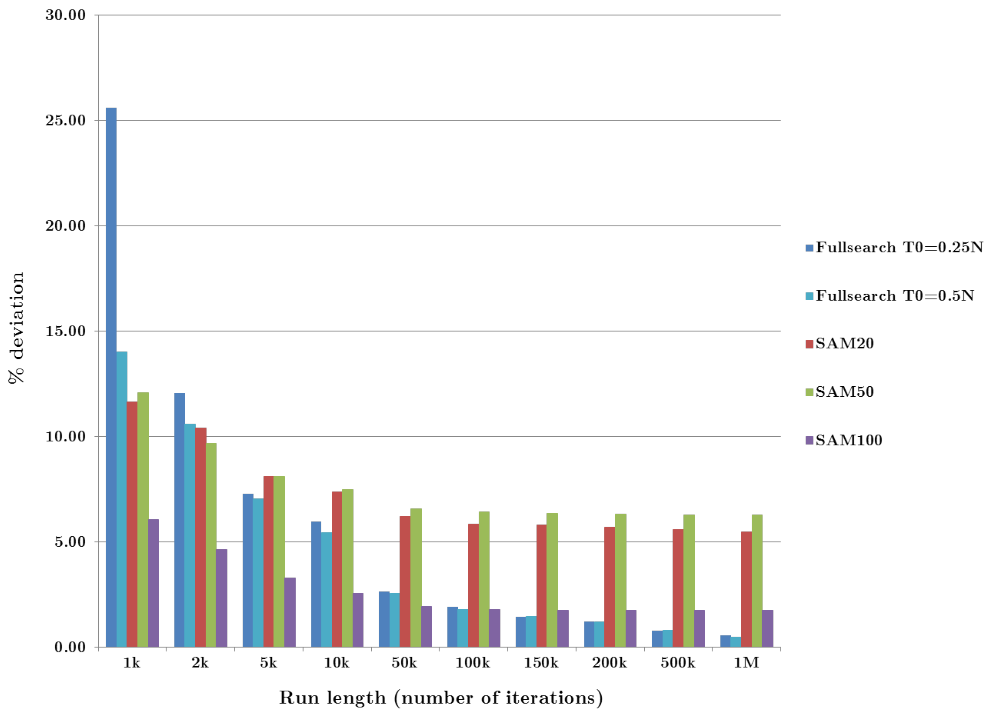


(d) Dataset 13 with 500 inner samples

Figure 5.7: Histogram showing the distribution produced by MH with 50, 100, 200 and 500 inner samples, dataset 13. Higher numbers of inner samples result in more concentrated distributions



(a) Average values



(b) Maximum values

FIGURE 5.8: Histogram of the results of SAM, taken over all problem sets, shown for SAM with 20 MH sections (SAM20), 50 sections and 100 sections (SAM50 and SAM100). The results of basic SA for two different starting temperatures are also shown for ease of comparison.

Chapter 6

Conclusion

The contributions of this work have been to investigate, in detail, the performance of Simulated Annealing in solving the single machine scheduling problem and, more importantly, to present the first hybrid MH-SA algorithm for use in optimisation of scheduling problems. The hybrid algorithm known as SAM is a unique application method of Metropolis-Hastings Monte Carlo to a problem that has not been done in the literature before, and that recognises and uses specific properties of the search space in solving this particular problem. It also relies on a new perspective of the search space. Despite its ability to specialise to the characteristics of the search space, it is also a general method applicable to a wide range of different problems and objectives. The performance on problems with different criteria is left as future work.

In Chapter 1, we give background relevant to the scheduling problem, define the specific problem that is the focus of this work, and present the hypotheses we aim to investigate and some assumptions we make to narrow the focus. We motivate the importance of investigating the problem by giving pertinent examples of applications where the particular scheduling problem could be useful. We also motivate why we have selected SA as the locus of concern. Important theoretical background on scheduling, computational complexity, Bayesian inference, Markov Chains and ordinary and MCMC as well as SA are presented in Chapter 2. Other related work is the subject of Chapter 3. In Chapter 4, we present the models by which our algorithm functions, prove why the section-based approach is reasonable and useful, and formulate the algorithms in detail. Some implementation details are also provided in this chapter, including consideration that had to be made in the generation of the many pseudo-random numbers that are required, and computational complexity of the algorithms are analysed. We find that the complexity of SAM is $O(n(N_M r + N_S) + m)$ while basic SA has a complexity of $O(nN_S)$ for n the number of jobs. We also contrast this much improved complexity with that of a complete search of all possible schedule permutations, $O(n \cdot n!)$ showing the remarkable improvement in complexity that can be achieved. Experimental results are presented in Chapter 5 showing trade-offs between different methods, computational load and solution quality.

The single machine scheduling problem for minimising weighted earliness and tardiness is NP-hard and requires a solution algorithm to scour a very large and turbulent search terrain that contains all the possible feasible permutations of jobs i.e. all possible schedules. As has been emphasised, with as few as ten jobs, the search space contains over 3.6 million points. In typical situations where the number of jobs is considerably greater, this search space is almost impossibly large. To illustrate we mentioned for 50 jobs the size is of the order 10^{64}. We have shown in Chapter 3 that previous literature has not provided a sufficient way of pruning the large search space in a generally applicable way, although researchers have investigated the performance of Simulated Annealing in solving the problem, as well as several other meta-heuristic approaches.

For this reason, we have presented the hybrid SAM algorithm which views the search space as a discrete piecewise-defined distribution dividing the large search space of permutations into a number of sections and defining likelihood values per section, which represent the likelihood of that section containing an optimal schedule. The details of the discrete distribution on sections are discovered using Metropolis-Hastings sampling. In this interpretation of MH sampling, the search space is divided into a specific number of sections, and the states to be sampled are sections of the search space and sampling states involves a process of inner sampling using a uniform distribution within the sample section state. This method chooses a likely section and so enables the large search space to be pruned to a manageable size so that SA is able to provide a similarly accurate solution schedule in a shorter run and fewer iterations. This new hybrid algorithm along with applicable assumptions is formulated and presented in Chapter 4.

The results of Chapter 5, in which the algorithms were run on thirteen different datasets, show that similar results to basic SA can be achieved by the hybrid SAM algorithm in much shorter run times to achieve the same percentage deviation from the optimum, even considering the uncertainty introduced in the MH step and the possibility that an incorrect section is chosen using this approach. We investigated the effect of starting temperature on the results of basic SA and found no statistically significant difference between using a temperature of 90720 and 1814400 (a quarter and a half the size of the search space, respectively). It is shown that between 150 000 and 200 000 iterations are required to get a solution with no more than 2% deviation from the optimum 99% of the time. In contrast, SAM can provide deviations of less than 2% in less than 1000 iterations in the case of 20 sections, and 2% deviation in less than 2000 iterations for

50 sections but only in some instances. If the coarse MH algorithm is run for 1000 iterations, reducing the search space by a factor of 20 and then running SA on the reduced search space for 10 000 iterations, a similar 5% deviation can be achieved as running SA on the full search space for 50 000 iterations. The improvement achieved is reducing the run time by a factor of 4.5, particularly useful for larger task sizes.

Using SAM, however, it is not possible to guarantee a 2% deviation 99% of the time as the results are so varied according to the performance of the Metropolis-Hastings sampling. When running time is of primary concern and some percentage deviation of up to 20% is acceptable, improved results can be achieved by SAM than by basic SA for low iteration counts (generally 1000 iterations and up to 100 000 iterations). There is a point, however, where the two cross over and it is more effective to run basic SA longer than to use SAM, to ensure lower deviations. Where longer runs are acceptable, the percentage deviation of basic SA tends to be considerably lower than that of SAM. We can conclude that, in general, a good implementation of SA can in fact produce good results regardless, even when the search space is very large. It must be noted, though, that it is possible that in the case of very large search space, such as when finding optima for 50 or 100-job schedules or larger, this search space may result in unacceptably long run times and SAM may become preferable in such a situation. Further experimentation with more jobs is required to verify this behaviour.

While simulation results have not shown conclusive superiority of our new hybrid algorithm and the unique implementation approach of MH sampling in the solution of the single machine weighted earliness-tardiness problem, we contend that the investigation was useful and provided a new approach to pruning the search space. We can accept the alternate hypothesis that a combination of Simulated Annealing and Metropolis-Hastings MCMC pre-sampling (to prune the search space) can be found that reduces the running time of SA (on the full search space), and makes a remarkable improvement on brute force search. Further experimentation may reveal clear situations in which our approach is superior and provide more justification for its usefulness. We also believe that this pre-sampling approach may be hybridised with other algorithms in other application scenarios and other problems in a useful way and further work in this field may have some utility. More experimentation on larger and more varied problems would be a good next step on from this work to verify our findings further, and especially to compare more rigorously our results with other hybrid and existing algorithms using the same problem sets. Another useful next step would be to use different sorting methods according to different distance metrics such as using Steinhaus–Johnson–Trotter or Heap's algorithms. These are likely to produce more clustered and less uniformly distributed cost terrains and may improve the SAM algorithm's performance. Implementation in real-world applications would also be very useful for further development and evaluation.

Appendix A

Complexity Classes

The references used here are the books by Arora and Barak [126] and Cormen et al. [127]. The goal of this section is to give just enough background about mathematical and computing topics that are essential to the research reported on in this document. This means it is, by no means, a formal or complete handling of any of the topics as this can be found elsewhere.

This research is primarily concerned with finding low complexity algorithms to solve problems in the category NP-hard or NP-complete. For this reason, some relevant formal definitions are presented. In Computer Science, problems are divided into classes of complexity depending on how difficult they are to solve.

Definition: A complexity class is a set of functions that can be computed within a given resource [126, 127].

A.0.1 Big-Oh notation

Definition: If f and g are two functions from $\mathbb{N}$ to $\mathbb{N}$ then we say that

i) $f(n) = O(g(n))$ if there exists a constant c such that $f(n) \leq c.g(n)$ for every sufficiently large n

ii) $f(n) = \Omega(g(n))$ if $g = O(f)$ or, equivalently $f(n) \geq c.g(n)$ for sufficiently large n

iii) $f(n) = O(g(n))$ if and only if $f = O(g)$ and $g = O(f)$ (i.e. $f(n) = \Omega(g(n))$)

The above statement also implies that the bound is tight.

iv) $f = O(g)$ if for every $\epsilon > 0, f(n) \leq \epsilon.g(n)$ for every sufficiently large n

v) $f = w(g)$ if $g = O(f)$

Statement i is most relevant to this work.

A.0.2 P, NP and NP-complete

Roughly speaking, the class of problems P are those for which an answer can be found in polynomial time.

An English definition is given instead of a Mathematical one for clarity (that is, to prevent the use of confusing notation). NP is the abbreviation of non-deterministic polynomial-time, and was originally defined in terms of a Non-deterministic Turing Machine (NDTM). A NDTM is one with two transition functions instead of one and at each transition point the machine makes an arbitrary choice which of these two functions is applied. This means that there exists a sequence of (non-deterministic) choices that results in the machine outputting 1 (YES) for input x. Specifically, a decision problem is in NP if, given an input x, we can easily verify that x is a YES instance of the problem if we are given the polynomial-size solution for x that certifies this fact. i.e. the YES answer can be accepted in polynomial time by a non-deterministic Turing machine [1]. However, no known polynomial-time algorithm exists to find the solution. Less formally, we can verify the solution easily but we cannot find the solution easily. The scheduling problem where the objective is to minimise weighted earliness and tardiness belongs to the class NP.

[1]A.M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem"[128]

Appendix B

Problem instances

Table B.1 gives all the parameter values for the datasets generated for, and used in, our experiments. Job processing times, due window start and end times, and earliness and tardiness penalties.

Table B.1: Problem Instances

Problem no. ***T, RDD***	**Processing times**	**Due window start**	**Due window end**	**Earliness penalty**	**Tardiness penalty**
0 **(0.1, 0.4)**	72.90	406.64	463.80	0.58	1.28
	91.62	521.51	543.23	0.78	4.02
	59.84	414.53	450.04	0.91	6.86
	52.02	520.01	563.49	0.29	7.57
	97.54	625.91	670.31	0.22	8.85
	66.49	427.94	465.56	0.34	7.04
	53.34	426.07	435.83	0.77	8.44
	79.06	562.64	605.51	0.49	1.87
	8.34	591.18	639.20	0.75	5.16
	33.66	621.71	649.90	0.98	2.60
1 **(0.1, 0.8)**	30.49	505.15	539.99	0.33	7.51
	33.87	601.26	602.52	0.39	7.13
	70.00	479.79	503.76	0.55	2.03
	85.98	437.42	446.97	0.18	1.63
	37.44	491.97	495.16	0.11	3.88
	59.86	536.28	580.34	0.61	1.90
	74.06	327.04	332.66	0.16	2.21
	85.89	458.38	489.42	0.46	7.01
	25.82	251.49	285.37	0.54	2.58
	22.83	326.00	347.29	0.38	3.11

Continued on next page

Problem no. ***T, RDD***	**Processing times**	**Due window start**	**Due window end**	**Earliness penalty**	**Tardiness penalty**
2 **(0.1, 1.2)**	36.09	212.47	213.60	0.86	3.67
	41.97	379.04	403.46	0.92	6.59
	86.87	718.76	771.93	0.18	7.96
	67.67	780.57	820.07	0.64	7.43
	88.43	479.93	507.13	0.23	5.88
	35.52	203.97	239.28	0.25	7.20
	78.83	282.45	313.73	0.68	1.36
	87.64	689.65	693.29	0.62	2.60
	6.44	482.11	511.40	0.10	2.31
	15.05	523.86	575.48	0.75	3.28
3 **(0.5, 0.4)**	72.90	160.72	217.88	0.58	1.28
	91.62	275.59	297.31	0.78	4.02
	59.84	168.61	204.12	0.91	6.86
	52.02	274.09	317.57	0.29	7.57
	97.54	380.00	424.39	0.22	8.85
	66.49	182.02	219.64	0.34	7.04
	53.34	180.15	189.91	0.77	8.44
	79.06	316.72	359.59	0.49	1.87
	8.34	345.26	393.28	0.75	5.16
	33.66	375.79	403.98	0.98	2.60
4 **(0.5, 0.8)**	30.49	294.65	329.50	0.33	7.51
	33.87	390.77	392.03	0.39	7.13
	70.00	269.29	293.26	0.55	2.03
	85.98	226.93	236.47	0.18	1.63
	37.44	281.47	284.67	0.11	3.88
	59.86	325.78	369.84	0.61	1.90
	74.06	116.55	122.16	0.16	2.21
	85.89	247.88	278.93	0.46	7.01
	25.82	41.00	74.87	0.54	2.58
	22.83	115.50	136.79	0.38	3.11
5 **(0.3, 0.8)**	72.00	164.28	221.00	0.58	1.28
	91.00	374.66	396.21	0.78	4.02
	59.00	169.20	204.44	0.91	6.86
	52.00	382.47	425.61	0.29	7.57
	97.00	593.08	637.13	0.22	8.85
	66.00	196.86	234.19	0.34	7.04
	53.00	179.32	189.01	0.77	8.44
	79.00	466.76	509.30	0.49	1.87
	8.00	525.95	573.60	0.75	5.16
	33.00	576.70	604.67	0.98	2.60

Continued on next page

Problem no. ***T, RDD***	**Processing times**	**Due window start**	**Due window end**	**Earliness penalty**	**Tardiness penalty**
	30.00	407.25	441.68	0.33	7.51
	33.00	521.81	523.07	0.39	7.13
	70.00	374.58	398.27	0.55	2.03
	85.00	320.47	329.91	0.18	1.63
6	37.00	387.06	390.23	0.11	3.88
(0.3, 1.0)	59.00	446.84	490.38	0.61	1.90
	74.00	183.64	189.20	0.16	2.21
	85.00	349.01	379.69	0.46	7.01
	25.00	93.82	127.30	0.54	2.58
	22.00	184.29	205.34	0.38	3.11
	36.00	102.52	103.65	0.86	3.67
	41.00	267.41	291.59	0.92	6.59
	86.00	603.71	656.34	0.18	7.96
	67.00	664.89	703.99	0.64	7.43
7	88.00	367.28	394.21	0.23	5.88
(0.3, 1.2)	35.00	94.11	129.07	0.25	7.20
	78.00	171.79	202.77	0.68	1.36
	87.00	574.89	578.50	0.62	2.60
	6.00	369.44	398.44	0.10	2.31
	15.00	410.78	461.87	0.75	3.28
	94.27	409.23	427.00	2.90	5.25
	33.09	712.38	738.00	5.30	9.60
	38.40	105.83	139.13	3.39	6.14
	48.76	420.42	437.00	0.74	1.33
8	61.55	504.35	550.33	4.55	8.23
(0.4, 0.6)	11.35	505.10	536.02	2.23	4.03
	30.75	632.13	676.69	2.10	3.80
	79.52	8.66	39.12	3.14	5.69
	47.34	43.68	47.35	3.45	6.23
	25.51	385.52	406.11	3.74	6.77
	1.00	172.67	179.33	0.80	2.00
	76.00	298.63	319.35	2.00	5.00
	54.00	264.56	274.98	1.20	3.00
	5.00	169.18	199.38	2.80	7.00
9	68.00	274.96	316.15	4.00	10.00
(0.4, 0.6)	39.00	231.83	255.16	2.40	6.00
	84.00	321.01	323.49	0.40	1.00
	6.00	173.52	197.30	2.40	6.00
	68.00	293.46	294.79	0.40	1.00
	39.00	241.54	245.42	0.40	1.00

Continued on next page

Problem no. ***T, RDD***	**Processing times**	**Due window start**	**Due window end**	**Earliness penalty**	**Tardiness penalty**
10 **(0.4, 0.6)**	89.00	385.71	438.72	4.00	10.00
	4.00	217.64	237.33	1.60	4.00
	76.00	379.32	388.90	0.80	2.00
	16.00	249.67	254.53	0.40	1.00
	65.00	351.63	367.85	1.20	3.00
	60.00	338.39	358.24	1.60	4.00
	58.00	341.56	346.52	0.40	1.00
	90.00	394.64	433.22	2.80	7.00
	11.00	236.57	245.09	0.80	2.00
	78.00	372.22	404.58	2.40	6.00
11 **(0.4, 0.6)**	68	374.31	403.90	2.40	6.00
	47.00	325.17	354.38	2.00	5.00
	15.00	243.99	285.43	3.20	8.00
	68.00	363.61	411.83	3.60	9.00
	91.00	419.23	466.89	3.60	9.00
	64.00	363.93	395.04	2.40	6.00
	78.00	386.97	435.89	3.60	9.00
	89.00	413.12	461.48	3.60	9.00
	26.00	285.35	294.77	0.80	2.00
	34.00	298.67	322.67	2.00	5.00
12 **(0.4, 0.6)**	32.00	267.36	288.20	1.60	4.00
	60.00	323.41	349.67	2.00	5.00
	15.00	228.14	255.87	2.40	6.00
	90.00	382.39	417.94	2.80	7.00
	85.00	376.51	401.90	2.00	5.00
	98.00	405.19	428.14	2.00	5.00
	32.00	258.99	295.34	2.80	7.00
	22.00	232.87	279.18	3.60	9.00
	6.23	1.00	2.00	2.00	6.00
	6.77	1.00	6.00	5.00	3.00

Appendix C

Results

C.1 Simulated Annealing base case

Table C.1: Summary of SA results

	Temp	**907200**				**1814400**				
	N	***Max %***	***Ave %***	***Min %***	***Var***	***Max %***	***Ave %***	***Min %***	***Var***	***t-test***
0	1k	24.60	3.47	0.32	29.45	3.72	1.70	0.37	0.57	T
	2k	3.02	1.54	0.11	0.33	2.25	1.44	0.41	0.21	F
	5k	2.23	1.02	0.29	0.17	1.99	1.03	0.31	0.16	F
	10k	2.05	0.74	0.12	0.16	1.47	0.81	0.00	0.11	F
	50k	0.75	0.37	0.00	0.04	0.73	0.38	0.00	0.04	F
	100k	0.68	0.25	0.00	0.03	0.52	0.23	0.00	0.02	F
	150k	0.42	0.18	0.00	0.02	0.63	0.22	0.00	0.03	F
	200k	0.42	0.17	0.00	0.01	0.44	0.16	0.00	0.02	F
	500k	0.29	0.07	0.00	0.01	0.29	0.07	0.00	0.00	F
	1M	0.18	0.05	0.00	0.62	0.08	0.03	0.00	0.00	T
1	1k	20.55	9.70	0.34	25.48	18.31	9.34	1.37	17.99	F
	2k	14.72	6.57	1.17	14.86	16.77	6.64	1.17	12.86	F
	5k	11.43	4.37	0.31	5.80	9.19	4.04	0.34	5.30	F
	10k	6.64	2.80	0.03	3.29	9.52	3.03	0.03	3.28	F
	50k	2.28	0.88	0.00	0.36	2.28	1.04	0.00	0.40	F
	100k	2.23	0.70	0.00	0.24	1.42	0.57	0.00	0.19	F
	150k	1.66	0.35	0.00	0.17	1.35	0.51	0.00	0.15	T
	200k	1.19	0.38	0.00	0.14	1.19	0.39	0.00	0.11	F

Continued on next page

	N	***Max %***	***Ave %***	***Min %***	***Var***	***Max %***	***Ave %***	***Min %***	***Var***	***t-test***
	500k	0.52	0.12	0.00	0.01	0.52	0.10	0.00	0.03	F
	1M	0.31	0.04	0.00	0.01	0.08	0.03	0.00	0.01	F
2	1k	36.26	9.98	0.84	34.64	15.40	9.70	3.71	9.40	F
	2k	20.32	8.42	1.58	13.27	15.32	7.88	1.78	8.72	F
	5k	10.74	5.74	0.61	4.37	11.54	5.98	1.37	4.52	F
	10k	9.37	4.68	0.63	2.67	8.31	4.87	1.10	2.42	F
	50k	4.78	2.90	0.74	1.00	5.49	2.72	0.00	1.98	F
	100k	3.97	2.00	0.15	1.05	3.75	1.79	0.00	1.06	F
	150k	3.05	1.49	0.00	0.64	3.57	1.49	0.00	0.87	F
	200k	3.13	1.16	0.00	0.77	2.82	1.10	0.00	0.59	F
	500k	1.99	0.65	0.00	0.26	1.99	0.58	0.00	35.87	F
	1M	1.93	0.42	0.00	0.18	1.58	0.40	0.00	0.16	F
3	1k	63.54	9.98	0.39	82.87	17.09	8.15	1.36	10.92	F
	2k	13.12	5.86	0.15	10.67	16.89	6.12	0.00	13.25	F
	5k	8.08	4.44	0.15	3.13	8.23	4.48	0.80	2.79	F
	10k	11.04	3.27	0.24	3.20	5.89	3.13	0.24	2.03	F
	50k	3.50	1.19	0.00	0.91	3.02	1.16	0.00	0.78	F
	100k	2.56	1.05	0.00	0.53	1.97	0.86	0.00	0.36	F
	150k	1.98	0.67	0.00	0.45	2.37	0.64	0.00	0.38	F
	200k	1.36	0.44	0.00	0.17	1.44	0.40	0.00	0.20	F
	500k	1.24	0.17	0.00	0.07	1.08	0.12	0.00	0.03	F
	1M	0.36	0.04	0.00	0.01	0.24	0.02	0.00	0.00	F
4	1k	33.09	12.68	1.65	54.80	28.89	10.70	3.12	27.01	F
	2k	21.65	9.14	2.64	14.97	16.14	7.91	2.09	13.27	F
	5k	13.42	5.67	0.66	8.49	11.69	5.45	1.06	7.16	F
	10k	10.85	4.10	0.00	4.56	8.91	4.46	0.88	4.47	F
	50k	4.77	2.17	0.40	1.01	4.69	2.09	0.00	1.11	F
	100k	3.74	1.35	0.00	0.71	3.01	1.39	0.00	0.47	F
	150k	2.43	1.23	0.15	0.37	2.35	1.10	0.00	0.36	F
	200k	1.91	0.97	0.00	0.22	1.72	0.90	0.00	0.23	F
	500k	1.47	0.56	0.00	0.13	1.28	0.47	0.00	0.14	F
	1M	1.28	0.39	0.00	0.11	1.06	0.35	0.00	0.09	F

Continued on next page

	N	***Max %***	***Ave %***	***Min %***	***Var***	***Max %***	***Ave %***	***Min %***	***Var***	***t-test***
5	1k	14.66	8.47	4.73	9.84	14.65	7.62	2.32	11.40	F
	2k	12.54	7.12	1.49	8.53	9.78	5.72	1.44	4.42	F
	5k	7.28	4.11	0.81	2.78	7.49	4.65	2.69	2.16	F
	10k	6.02	3.83	1.41	1.37	7.13	3.60	1.28	2.26	F
	50k	3.85	1.84	0.52	0.83	2.98	1.84	0.26	0.59	F
	100k	2.37	1.27	0.00	0.53	2.26	1.19	0.26	0.46	F
	150k	2.05	1.25	0.00	0.22	2.02	0.92	0.00	0.42	F
	200k	1.74	0.75	0.00	0.22	1.76	0.81	0.00	0.24	F
	500k	1.00	0.32	0.00	0.09	1.20	0.41	0.00	0.14	F
	1M	0.63	0.25	0.00	0.04	0.88	0.27	0.00	0.07	F
6	1k	3.97	2.48	0.44	1.05	4.84	2.73	0.47	1.37	F
	2k	4.38	2.11	0.61	0.77	4.12	2.23	0.94	0.80	F
	5k	3.06	1.60	0.69	0.46	2.38	1.53	0.16	0.39	F
	10k	1.72	1.02	0.00	0.16	2.13	1.30	0.59	0.17	F
	50k	1.35	0.58	0.03	0.11	1.10	0.68	0.00	0.09	F
	100k	0.74	0.41	0.00	0.05	0.80	0.31	0.00	0.07	F
	150k	0.92	0.36	0.00	0.06	0.78	0.30	0.00	0.04	F
	200k	0.64	0.30	0.00	0.05	0.71	0.30	0.00	0.05	F
	500k	0.44	0.14	0.00	0.02	0.49	0.22	0.00	0.03	F
	1M	0.19	0.05	0.00	0.00	0.32	0.09	0.00	0.01	F
7	1k	26.18	8.24	1.12	41.37	10.76	4.99	3.11	2.99	T
	2k	10.59	4.16	0.97	4.87	5.81	3.53	1.43	2.12	F
	5k	4.93	2.68	1.16	0.82	4.65	2.48	0.52	1.21	F
	10k	4.13	1.76	0.53	0.77	3.40	1.92	0.40	0.55	F
	50k	1.60	0.91	0.25	0.20	2.10	0.80	0.17	0.24	F
	100k	1.63	0.65	0.12	0.14	1.55	0.55	0.00	0.14	F
	150k	1.13	0.45	0.00	0.07	0.72	0.38	0.15	0.03	F
	200k	0.98	0.36	0.00	0.05	1.04	0.42	0.00	0.07	F
	500k	0.66	0.22	0.00	0.03	0.41	0.18	0.00	0.01	F
	1M	0.25	0.13	0.00	0.01	0.26	0.11	0.00	0.01	F

Continued on next page

	N	***Max %***	***Ave %***	***Min %***	***Var***	***Max %***	***Ave %***	***Min %***	***Var***	***t-test***
8	1k	34.76	13.02	1.88	60.63	22.97	11.36	1.71	27.57	F
	2k	25.87	9.60	0.32	51.45	25.38	9.82	0.68	46.09	F
	5k	14.31	5.32	1.01	13.08	12.12	5.79	1.27	10.01	F
	10k	8.65	2.82	0.34	3.54	8.19	2.89	0.33	3.13	F
	50k	4.70	1.52	0.00	1.23	3.00	1.46	0.33	0.71	F
	100k	2.08	0.99	0.00	0.26	2.37	1.20	0.33	0.34	F
	150k	1.84	0.85	0.00	0.22	1.90	0.97	0.27	0.24	F
	200k	1.88	0.93	0.32	0.23	1.21	0.53	0.00	0.09	T
	500k	0.99	0.42	0.00	0.07	1.26	0.60	0.00	0.10	F
	1M	0.86	0.28	0.00	0.05	0.86	0.28	0.00	0.05	F
9	1k	8.43	2.55	0.34	3.33	6.09	2.64	0.09	2.19	F
	2k	5.69	1.77	0.35	1.32	3.42	1.51	0.35	0.94	F
	5k	2.71	1.07	0.15	0.36	1.97	1.14	0.07	0.31	F
	10k	1.41	0.58	0.03	0.16	1.57	0.62	0.01	0.18	F
	50k	0.73	0.29	0.00	0.04	0.43	0.21	0.01	0.01	F
	100k	0.39	0.14	0.00	0.01	0.32	0.12	0.00	0.01	F
	150k	0.33	0.11	0.01	0.01	0.26	0.09	0.01	0.00	F
	200k	0.32	0.11	0.00	0.01	0.41	0.10	0.00	0.01	F
	500k	0.18	0.05	0.00	0.00	0.12	0.05	0.00	0.00	F
	1M	0.09	0.02	0.00	0.00	0.07	0.03	0.00	0.00	F
10	1k	17.27	7.38	2.16	14.94	20.60	10.60	3.56	22.91	T
	2k	13.44	6.67	2.90	8.41	10.77	7.05	1.90	7.47	F
	5k	8.97	5.08	2.40	3.06	9.55	4.19	1.69	4.18	F
	10k	6.77	3.54	1.63	1.97	7.42	4.03	0.48	3.64	F
	50k	2.88	1.80	0.48	0.40	3.96	1.88	0.48	0.92	F
	100k	2.48	1.30	0.00	0.57	2.90	1.48	0.48	0.58	F
	150k	1.86	0.85	0.00	0.23	1.65	0.88	0.00	0.22	F
	200k	1.38	0.88	0.48	0.11	1.65	0.86	0.00	0.20	F
	500k	0.97	0.37	0.00	0.10	1.84	0.41	0.00	0.19	F
	1M	1.14	0.32	0.00	0.10	0.60	0.24	0.00	0.06	F

Continued on next page

	N	***Max %***	***Ave %***	***Min %***	***Var***	***Max %***	***Ave %***	***Min %***	***Var***	***t-test***
11	1k	15.31	8.06	2.10	15.29	14.79	7.08	0.99	13.60	F
	2k	10.31	5.37	1.15	5.32	9.53	4.83	0.00	5.69	F
	5k	6.71	4.16	2.03	1.83	7.18	4.07	1.77	2.50	F
	10k	7.10	2.57	0.40	2.40	5.64	2.13	0.00	2.07	F
	50k	3.08	1.11	0.12	0.49	1.81	0.64	0.12	0.25	T
	100k	1.37	0.56	0.00	0.18	1.70	0.66	0.00	0.21	F
	150k	1.01	0.48	0.00	0.13	1.02	0.32	0.00	0.11	F
	200k	1.02	0.34	0.00	0.08	1.30	0.42	0.00	0.10	F
	500k	0.47	0.14	0.00	0.02	0.26	0.09	0.00	0.01	F
	1M	0.19	0.06	0.00	0.01	0.40	0.06	0.00	0.01	F
12	1k	13.15	7.29	1.08	13.39	14.71	9.25	4.29	7.80	F
	2k	10.64	4.59	0.13	7.34	9.83	5.23	0.97	5.46	F
	5k	6.03	3.89	1.93	1.34	8.56	3.56	1.41	2.94	F
	10k	6.50	2.96	0.22	2.49	5.51	2.76	0.22	1.79	F
	50k	2.56	1.41	0.13	0.54	3.60	1.15	0.13	0.93	F
	100k	2.37	0.84	0.00	0.60	2.32	0.95	0.08	0.41	F
	150k	1.35	0.54	0.00	0.18	1.65	0.49	0.00	0.20	F
	200k	1.19	0.32	0.00	0.08	1.02	0.45	0.13	0.08	F
	500k	0.68	0.20	0.00	0.03	0.47	0.22	0.00	0.02	F
	1M	0.29	0.09	0.00	0.01	0.23	0.08	0.00	0.00	F
Avg	1k	25.60	7.84	1.25	168.78	14.04	6.98	1.91	47.72	
	2k	12.07	5.31	1.00	39.80	10.59	5.06	0.95	33.42	
	5k	7.28	3.57	0.88	13.53	7.04	3.51	0.96	12.01	
	10k	5.98	2.52	0.40	9.77	5.44	2.57	0.40	7.87	
	50k	2.67	1.23	0.19	2.00	2.55	1.16	0.11	2.00	
	100k	1.93	0.83	0.02	1.14	1.80	0.82	0.08	0.96	
	150k	1.46	0.64	0.01	0.64	1.46	0.60	0.03	0.70	
	200k	1.24	0.51	0.06	0.48	1.21	0.49	0.01	0.43	
	500k	0.79	0.25	0.00	0.21	0.81	0.25	0.00	0.26	
	1M	0.56	0.16	0.00	0.16	0.48	0.14	0.00	0.12	

C.2 Metropolis-Hastings simulations

C.2.1 Chi-squared test of significance

Table C.2: Dataset 0 MH: Summary for division of search space into 100 sections, part 1

Number of samples		**50**						**100**					
		number of iterations						***number of iterations***					
Section	*Ho*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*
0	0.2	0	0	0	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮	⋮
93	0.2	1	1	2	1	3	1	1	0	2	2	1	3
94	0.2	0	0	0	0	0	0	0	0	0	0	0	0
95	0.2	1	0	1	0	0	0	1	2	1	1	0	2
96	0.2	8	8	5	4	4	7	6	8	7	8	8	4
97	0.2	10	11	11	15	12	12	11	10	10	9	11	11
98	0.2	0	0	0	0	0	0	0	0	0	0	0	0
99	0.2	0	0	0	0	0	0	0	0	0	0	0	0
	p-value	4.9E-112	2.6E-131	3.61E-98	1.8E-186	2.7E-115	4.4E-139	9.6E-106	6E-114	1.3E-100	7.54E-97	2.6E-131	7.54E-97

Table C.3: Dataset 0 MH: Summary for division of search space into 100 sections, part 2

Number of samples		**200**						**500**					
		number of iterations						***number of iterations***					
Section	*Ho*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*
0	0.2	0	0	0	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
93	0.2	0	1	0	0	2	0	0	0	0	0	1	0
94	0.2	0	0	0	0	0	0	0	0	0	0	0	0
95	0.2	0	0	1	0	1	0	0	0	1	0	0	1
96	0.2	6	11	9	12	6	7	6	10	7	6	7	8
97	0.2	14	8	10	8	11	13	14	10	12	14	12	11
98	0.2	0	0	0	0	0	0	0	0	0	0	0	0
99	0.2	0	0	0	0	0	0	0	0	0	0	0	0
p-value		1.6E-176	2.6E-131	2E-127	8.6E-153	3.2E-108	1.2E-162	1.6E-176	6E-145	4.4E-139	1.6E-176	4.4E-139	2.6E-131

Table C.4: Dataset 0 MH: Summary for division of search space into 50 sections, part 2

Number of samples		**200**						**500**					
		number of iterations						***number of iterations***					
Section	*Ho*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*
1	0.4	0	0	0	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
46	0.4	0	0	0	1	0	1	0	0	0	0	0	2
47	0.4	0	2	2	1	2	0	1	0	0	0	0	0
48	0.4	20	18	18	18	18	19	19	20	20	20	20	18
49	0.4	0	0	0	0	0	0	0	0	0	0	0	0
50	0.4	0	0	0	0	0	0	0	0	0	0	0	0
	p-value	1.2E-204	9.8E-166	9.8E-166	1.2E-164	9.8E-166	4.2E-184	4.12E-184	1.2E-204	1.2E-204	1.2E-204	1.2E-204	9.8E-166

Table C.5: Dataset 0 MH: Summary for division of search space into 20 sections, part 1

Number of samples		**50**						**100**					
		number of iterations						***number of iterations***					
Section	*H0*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*
1	1	0	0	0	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
18	1	4	4	6	2	4	5	4	2	2	4	2	2
19	1	16	16	14	18	16	15	16	18	18	16	18	18
20	1	0	0	0	0	0	0	0	0	0	0	0	0
	p-value	1.2E-42	1.2E-42	1.4E-34	4.6E-54	1.2E-42	3.4E-38	1.2E-42	4.6E-54	4.6E-54	1.2E-42	4.6E-54	4.6E-54

Table C.6: Dataset 0 MH: Summary for division of search space into 20 sections, part 2

Number of samples		**200**						**500**					
		number of iterations						***number of iterations***					
Section	*H0*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*	*500*	*1000*	*2000*	*5000*	*10000*	*50000*
1	1	0	0	0	0	0	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
18	1	4	2	2	1	1	3	1	0	2	1	0	3
19	1	16	18	18	19	19	17	19	20	18	19	20	17
20	1	0	0	0	0	0	0	0	0	0	0	0	0
	p-value	1.2E-4	4.6E-54	4.6E-54	4.6E-61	4.6E-61	6.3E-48	4.6E-61	6.3E-69	4.6E-54	4.6E-61	6.3E-69	6.3E-48

C.2.2 Impact of iteration count

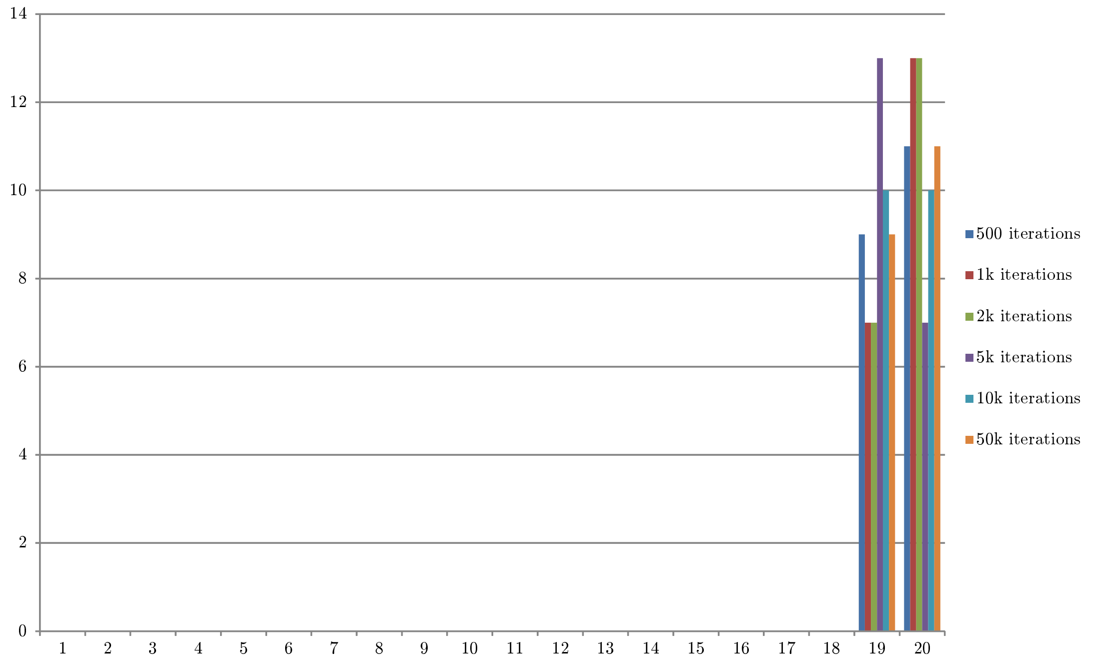


FIGURE C.1: The effect of different run lengths on the frequency distribution for 20 section divisions, 500 inner samples: dataset 1

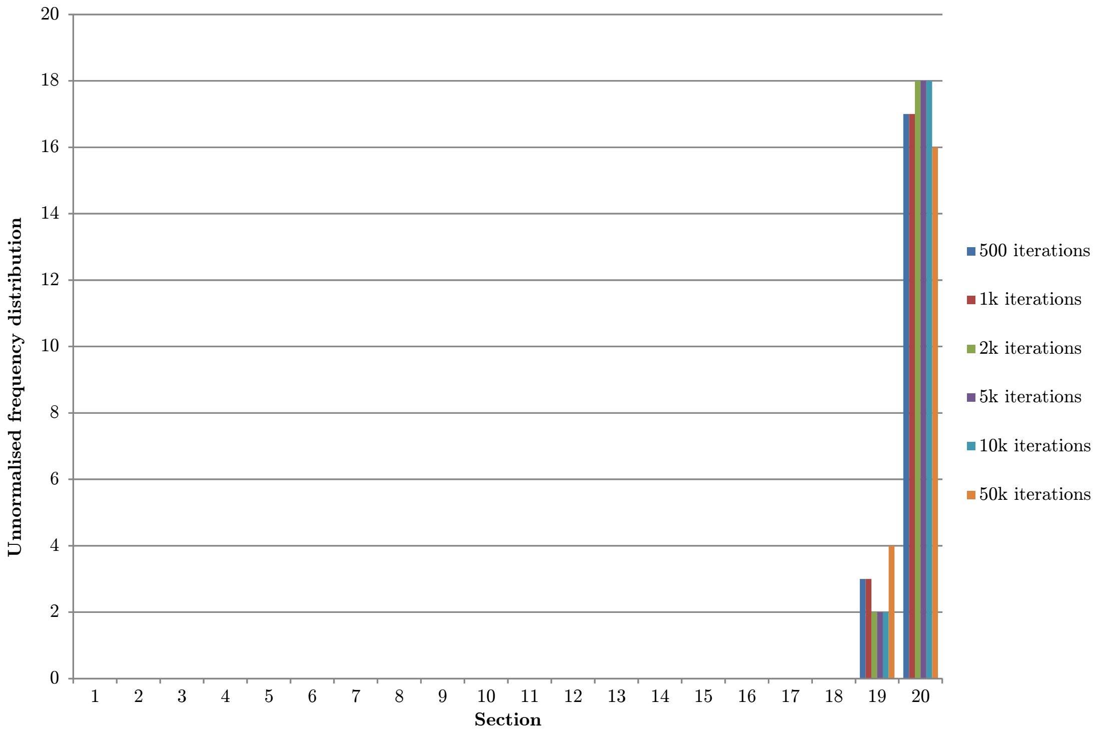


FIGURE C.2: The effect of different run lengths on the frequency distribution for 20 section divisions, 500 inner samples: dataset 2

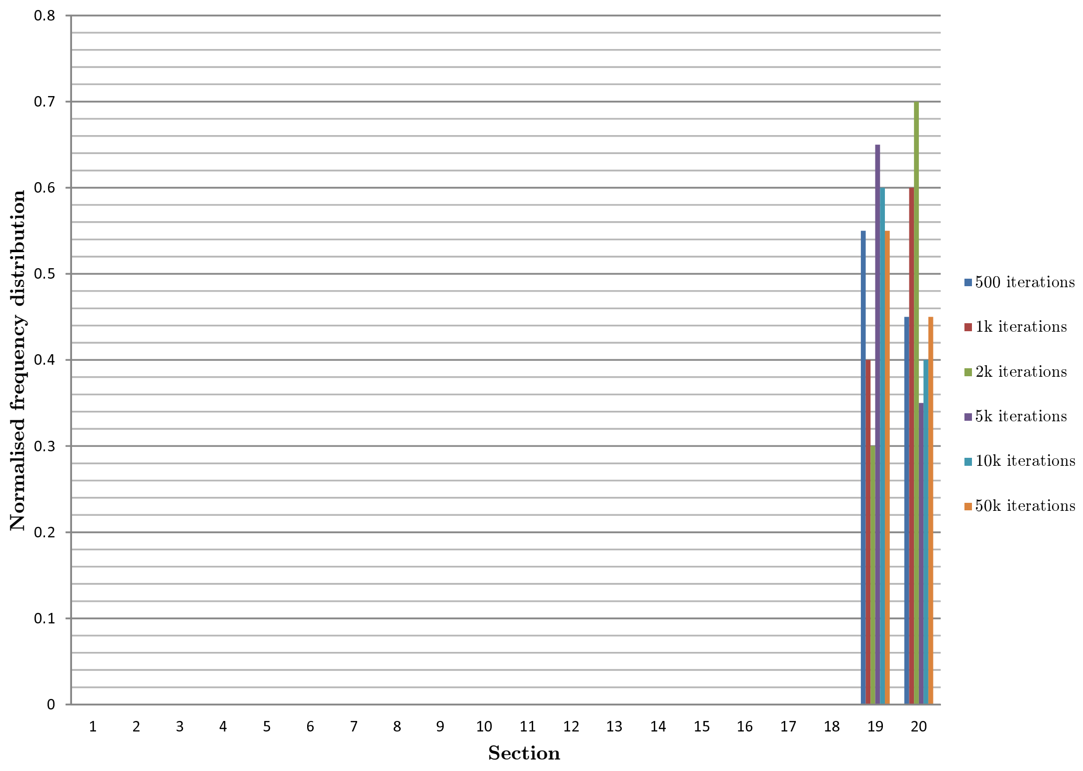


FIGURE C.3: The effect of different run lengths on the frequency distribution for 20 section divisions, 500 inner samples: dataset 3

C.3 Reduced search SA final results

Table C.7 shows maximum (Max), mean and minimum (Min) percentage deviations from the optimal value obtained in 50 runs of basic SA compared with the two-step SAM algorithm for division of search space into 20 (SAM20), 50 (SAM50), and 100 sections (SAM100). The columns headed $|t| > t_{crit}$ show T if the set of results were found to be statistically significantly different from basic SA by the t-test for significance, and F if the differences were not found to be significant. The iteration values represent the algorithms' run times. From the results, it is clear that for shorter runs, our modified SAM algorithm yields significantly better results than basic SA. The point at which basic SA starts to catch up in longer runs differs per dataset from 10 000 to 200 000. If this algorithm is to be deployed in network managers for scheduling packets in wireless networks, the time cost of runs longer than 10 000 iterations is not acceptable to provide QoS as network delays will be too long. In this situation, our SAM algorithm is clearly the preferred choice, as it can yield deviations as low as 0.15% in a mere 1000 iterations.

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			SAM20				SAM50				SAM100			
Dataset	**N**	***Max* %**	***Ave* %**	***Min* %**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**
0	**1k**	3.72	1.70	0.37	2.03	0.64	0.08	T	2.46	0.42	0.03	T	1.70	0.41	0.00	T
	2k	2.25	1.44	0.41	1.70	0.51	0.07	T	1.69	0.34	0.07	T	1.20	0.28	0.00	T
	5k	1.99	1.03	0.31	1.36	0.30	0.00	T	1.41	0.16	0.00	T	1.20	0.16	0.00	T
	10k	1.47	0.81	0.00	1.27	0.23	0.00	T	1.27	0.11	0.00	T	0.82	0.08	0.00	T
	50k	0.73	0.38	0.00	0.76	0.09	0.00	F	0.71	0.04	0.00	F	0.65	0.03	0.00	T
	100k	0.52	0.23	0.00	0.66	0.07	0.00	F	0.66	0.03	0.00	F	0.61	0.03	0.00	F
	150k	0.63	0.22	0.00	0.61	0.07	0.00	F	0.61	0.03	0.00	T	0.58	0.03	0.00	F
	200k	0.44	0.16	0.00	0.66	0.06	0.00	F	0.61	0.03	0.00	T	0.58	0.03	0.00	F
	500k	0.29	0.07	0.00	0.58	0.06	0.00	F	0.58	0.03	0.00	T	0.58	0.03	0.00	T
	1M	0.08	0.03	0.00	0.58	0.06	0.00	F	0.58	0.03	0.00	T	0.58	0.03	0.00	T
1	**1k**	18.31	9.34	1.37	12.95	4.68	0.00	T	4.97	2.15	0.03	T	6.57	1.86	0.00	T
	2k	16.77	6.64	1.17	11.76	3.80	0.03	T	3.88	1.57	0.00	T	5.28	1.09	0.00	T
	5k	9.19	4.04	0.34	10.33	3.00	0.00	T	2.41	0.81	0.00	T	3.39	0.73	0.00	T
	10k	9.52	3.03	0.03	7.20	2.50	0.00	F	1.40	0.56	0.00	T	1.68	0.57	0.00	T
	50k	2.28	1.04	0.00	6.08	2.15	0.00	T	0.31	0.40	0.00	T	1.14	0.52	0.00	T
	100k	1.42	0.57	0.00	6.05	2.12	0.00	T	0.31	0.40	0.00	F	1.14	0.52	0.00	T
	150k	1.35	0.51	0.00	6.05	2.13	0.00	T	0.31	0.40	0.00	F	1.14	0.52	0.00	T
	200k	1.19	0.39	0.00	6.05	2.12	0.00	T	0.31	0.40	0.00	T	1.14	0.52	0.00	T
	500k	0.52	0.10	0.00	6.05	2.12	0.00	T	0.31	0.40	0.00	T	1.14	0.52	0.00	T
	1M	0.08	0.03	0.00	6.05	2.12	0.00	T	0.31	0.40	0.00	T	1.14	0.52	0.00	T

Continued on next page

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			**SAM20**				**SAM50**				**SAM100**			
Dataset	**N**	***Max*** *%*	***Ave*** *%*	***Min*** *%*	***Max*** *%*	***Ave*** *%*	***Min*** *%*	$\lvert\mathbf{t}\rvert > \mathbf{t_{crit}}$	***Max*** *%*	***Ave*** *%*	***Min*** *%*	$\lvert\mathbf{t}\rvert > \mathbf{t_{crit}}$	***Max*** *%*	***Ave*** *%*	***Min*** *%*	$\lvert\mathbf{t}\rvert > \mathbf{t_{crit}}$
2	**1k**	15.40	9.70	3.71	11.83	4.41	0.15	T	8.61	3.55	0.15	T	8.86	3.30	0.00	T
	2k	15.32	7.88	1.78	16.45	3.95	0.20	T	8.35	2.85	0.61	T	6.61	2.57	0.00	T
	5k	11.54	5.98	1.37	7.55	2.67	0.15	T	5.79	1.79	0.00	T	5.40	1.60	0.00	T
	10k	8.31	4.87	1.10	6.53	1.89	0.00	T	5.30	1.56	0.00	T	4.78	1.27	0.00	T
	50k	5.49	2.72	0.00	4.77	1.00	0.00	T	2.97	0.83	0.00	T	2.91	0.77	0.00	T
	100k	3.75	1.79	0.00	3.17	0.78	0.00	F	2.71	0.72	0.00	T	2.43	0.70	0.00	T
	150k	3.57	1.49	0.00	2.97	0.67	0.00	F	2.43	0.69	0.00	F	2.23	0.68	0.00	T
	200k	2.82	1.10	0.00	2.71	0.61	0.00	F	2.43	0.68	0.00	F	2.23	0.68	0.00	F
	500k	1.99	0.58	0.00	2.71	0.58	0.00	F	2.23	0.67	0.00	F	2.23	0.68	0.00	F
	1M	1.58	0.40	0.00	2.23	0.56	0.00	T	2.23	0.67	0.00	T	2.23	0.68	0.00	T
3	**1k**	17.09	8.15	1.36	7.68	3.34	0.00	T	9.80	2.45	0.00	T	5.13	2.11	0.00	T
	2k	16.89	6.12	0.00	7.07	2.38	0.00	T	4.99	1.77	0.00	T	4.64	1.56	0.00	T
	5k	8.23	4.48	0.80	3.37	1.23	0.00	T	4.15	0.92	0.00	T	2.81	1.08	0.00	T
	10k	5.89	3.13	0.24	2.32	0.79	0.00	T	2.97	0.58	0.00	T	2.81	0.89	0.00	T
	50k	3.02	1.16	0.00	0.82	0.11	0.00	T	1.33	0.29	0.00	T	2.32	0.82	0.00	T
	100k	1.97	0.86	0.00	0.39	0.06	0.00	T	1.33	0.28	0.00	T	2.32	0.82	0.00	F
	150k	2.37	0.64	0.00	0.15	0.05	0.00	T	1.33	0.28	0.00	F	2.32	0.82	0.00	F
	200k	1.44	0.40	0.00	0.15	0.05	0.00	T	1.33	0.28	0.00	F	2.32	0.82	0.00	T
	500k	1.08	0.12	0.00	0.15	0.05	0.00	F	1.33	0.28	0.00	T	2.32	0.82	0.00	T
	1M	0.24	0.02	0.00	0.15	0.05	0.00	T	1.33	0.28	0.00	T	2.32	0.82	0.00	T

Continued on next page

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			**SAM20**				**SAM50**				**SAM100**			
Dataset	**N**	***Max* %**	***Ave* %**	***Min* %**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**
4	**1k**	28.89	10.70	3.12	10.45	4.32	0.40	T	13.65	3.96	0.66	T	8.12	3.07	0.00	T
	2k	16.14	7.91	2.09	5.87	2.31	0.40	T	6.34	2.88	3.12	T	7.98	2.20	0.00	T
	5k	11.69	5.45	1.06	2.71	1.40	0.15	T	5.46	1.98	0.00	T	3.21	1.32	0.00	T
	10k	8.91	4.46	0.88	2.16	0.91	0.00	T	4.84	1.50	0.00	T	2.44	1.02	0.00	T
	50k	4.69	2.09	0.00	1.28	0.34	0.00	T	3.26	0.96	0.00	T	1.50	0.65	0.00	T
	100k	3.01	1.39	0.00	0.55	0.18	0.00	T	3.12	0.91	0.00	F	1.50	0.63	0.00	T
	150k	2.35	1.10	0.00	0.55	0.14	0.00	T	3.12	0.88	0.00	F	1.47	0.62	0.00	T
	200k	1.72	0.90	0.00	0.40	0.05	0.00	F	3.12	0.87	0.00	T	1.47	0.63	0.00	F
	500k	1.28	0.47	0.00	0.00	0.00	0.00	T	3.08	0.87	0.00	T	1.47	0.62	0.00	T
	1M	1.06	0.35	0.00	0.00	0.00	0.00	T	3.08	0.87	0.00	T	1.47	0.62	0.00	T
5	**1k**	14.65	7.62	2.32	9.44	4.13	0.52	T	6.71	2.43	0.17	T	4.21	1.93	0.00	T
	2k	9.78	5.72	1.44	9.52	3.34	0.05	T	5.46	1.83	0.00	T	3.27	1.53	0.00	T
	5k	7.49	4.65	2.69	6.18	2.73	0.17	T	2.91	1.03	0.00	T	2.17	0.93	0.00	T
	10k	7.13	3.60	1.28	5.22	2.38	0.00	T	1.99	0.73	0.00	T	1.35	0.63	0.00	T
	50k	2.98	1.84	0.26	3.69	1.70	0.00	F	0.81	0.34	0.00	T	0.72	0.45	0.00	T
	100k	2.26	1.19	0.26	3.55	1.62	0.00	F	0.92	0.31	0.00	T	0.60	0.42	0.00	T
	150k	2.02	0.92	0.00	3.62	1.62	0.00	T	0.60	0.28	0.00	T	0.55	0.42	0.00	F
	200k	1.76	0.81	0.00	3.24	1.56	0.00	T	0.60	0.28	0.00	T	0.55	0.42	0.00	F
	500k	1.20	0.41	0.00	3.24	1.54	0.00	T	0.55	0.27	0.00	F	0.55	0.42	0.00	T
	1M	0.88	0.27	0.00	3.06	1.53	0.00	T	0.55	0.27	0.00	F	0.55	0.42	0.00	T

Continued on next page

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			**SAM20**				**SAM50**				**SAM100**			
Dataset	**N**	***Max* %**	***Ave* %**	***Min* %**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**
6	**1k**	4.84	2.73	0.47	2.51	1.14	0.03	T	2.25	1.26	0.16	T	3.41	0.77	0.00	T
	2k	4.12	2.23	0.94	1.77	0.83	0.03	T	1.92	0.96	0.00	T	3.03	0.65	0.00	T
	5k	2.38	1.53	0.16	1.58	0.62	0.00	T	1.56	0.79	0.03	T	2.57	0.40	0.00	T
	10k	2.13	1.30	0.59	1.00	0.45	0.03	T	1.21	0.64	0.00	T	2.26	0.30	0.00	T
	50k	1.10	0.68	0.00	0.60	0.15	0.00	T	0.69	0.36	0.00	T	1.88	0.13	0.00	T
	100k	0.80	0.31	0.00	0.33	0.06	0.00	T	0.63	0.31	0.00	F	1.73	0.13	0.00	F
	150k	0.78	0.30	0.00	0.16	0.03	0.00	T	0.61	0.30	0.00	F	1.64	0.13	0.00	F
	200k	0.71	0.30	0.00	0.19	0.02	0.00	T	0.61	0.30	0.00	F	1.64	0.12	0.00	T
	500k	0.49	0.22	0.00	0.03	0.00	0.00	T	0.60	0.29	0.00	T	1.64	0.12	0.00	T
	1M	0.32	0.09	0.00	0.03	0.00	0.00	T	0.60	0.29	0.00	T	1.64	0.12	0.00	T
7	**1k**	10.76	4.99	3.11	4.29	1.81	0.12	T	3.84	1.70	0.05	T	2.54	1.20	0.00	T
	2k	5.81	3.53	1.43	2.83	1.36	0.00	T	2.59	1.11	0.00	T	2.22	0.86	0.00	T
	5k	4.65	2.48	0.52	1.74	0.85	0.05	T	1.92	0.76	0.05	T	1.94	0.49	0.00	T
	10k	3.40	1.92	0.40	1.79	0.57	0.00	T	1.26	0.55	0.00	T	1.13	0.34	0.00	T
	50k	2.10	0.80	0.17	0.54	0.22	0.00	T	0.37	0.18	0.00	T	0.50	0.15	0.00	T
	100k	1.55	0.55	0.00	0.37	0.15	0.00	T	0.33	0.11	0.00	T	0.34	0.13	0.00	T
	150k	0.72	0.38	0.15	0.33	0.10	0.00	T	0.25	0.09	0.00	T	0.34	0.13	0.00	T
	200k	1.04	0.42	0.00	0.07	0.07	0.00	T	0.31	0.08	0.00	T	0.28	0.13	0.00	T
	500k	0.41	0.18	0.00	0.25	0.04	0.00	T	0.12	0.07	0.00	T	0.28	0.13	0.00	T
	1M	0.26	0.11	0.00	0.12	0.03	0.00	T	0.12	0.07	0.00	F	0.28	0.13	0.00	F

Continued on next page

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			**SAM20**				**SAM50**				**SAM100**			
Dataset	**N**	***Max* %**	***Ave* %**	***Min* %**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**
8	**1k**	22.97	11.36	1.71	7.94	3.14	0.54	T	9.38	3.66	0.27	T	12.74	2.56	0.00	T
	2k	25.38	9.82	0.68	4.93	2.19	0.00	T	7.27	2.82	0.00	T	6.87	1.87	0.27	T
	5k	12.12	5.79	1.27	3.86	1.57	0.27	T	5.25	1.99	0.00	T	6.45	1.49	0.00	T
	10k	8.19	2.89	0.33	3.88	1.21	0.27	T	4.41	1.62	0.00	T	4.31	1.21	0.00	T
	50k	3.00	1.46	0.33	1.29	0.57	0.00	T	3.96	1.29	0.00	F	3.48	0.87	0.00	F
	100k	2.37	1.20	0.33	0.91	0.34	0.00	T	3.33	1.18	0.00	F	3.41	0.81	0.00	F
	150k	1.90	0.97	0.27	1.21	0.29	0.00	T	3.33	1.14	0.00	T	3.47	0.80	0.00	T
	200k	1.21	0.53	0.00	0.91	0.28	0.00	T	3.31	1.14	0.00	T	3.31	0.79	0.00	T
	500k	1.26	0.60	0.00	0.60	0.16	0.00	T	3.31	1.12	0.00	T	3.31	0.79	0.00	T
	1M	0.86	0.28	0.00	0.27	0.13	0.00	T	3.31	1.12	0.00	T	3.31	0.79	0.00	T
9	**1k**	6.09	2.64	0.09	2.02	0.80	0.02	T	2.33	0.80	0.01	T	2.35	0.77	0.00	T
	2k	3.42	1.51	0.35	1.71	0.45	0.00	T	1.71	0.51	0.00	T	1.46	0.54	0.00	T
	5k	1.97	1.14	0.07	0.73	0.24	0.00	T	1.11	0.37	0.00	T	1.27	0.45	0.00	T
	10k	1.57	0.62	0.01	0.51	0.13	0.01	T	1.06	0.31	0.00	T	1.16	0.43	0.00	T
	50k	0.43	0.21	0.01	0.25	0.06	0.00	T	0.91	0.25	0.00	F	1.06	0.39	0.00	T
	100k	0.32	0.12	0.00	0.12	0.04	0.00	T	0.77	0.24	0.00	T	1.03	0.39	0.00	T
	150k	0.26	0.09	0.01	0.08	0.03	0.00	T	0.77	0.24	0.00	T	1.03	0.39	0.00	T
	200k	0.41	0.10	0.00	0.08	0.03	0.00	T	0.77	0.24	0.00	T	1.02	0.39	0.00	T
	500k	0.12	0.05	0.00	0.06	0.03	0.00	T	0.76	0.24	0.00	T	1.02	0.39	0.00	T
	1M	0.07	0.03	0.00	0.05	0.03	0.00	F	0.76	0.24	0.00	T	1.02	0.39	0.00	T

Continued on next page

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			**SAM20**				**SAM50**				**SAM100**			
Dataset	**N**	***Max* %**	***Ave* %**	***Min* %**	***Max* %**	***Ave* %**	***Min* %**	$\mathbf{\lvert t\rvert > t_{crit}}$	***Max* %**	***Ave* %**	***Min* %**	$\mathbf{\lvert t\rvert > t_{crit}}$	***Max* %**	***Ave* %**	***Min* %**	$\mathbf{\lvert t\rvert > t_{crit}}$
10	**1k**	20.60	10.60	3.56	11.58	4.75	0.80	T	19.12	3.52	0.04	T	7.04	2.26	0.00	T
	2k	10.77	7.05	1.90	9.78	4.11	0.49	T	8.97	2.41	0.00	T	3.78	1.69	0.00	T
	5k	9.55	4.19	1.69	7.55	3.17	0.04	T	4.93	1.60	0.00	T	3.55	1.23	0.00	T
	10k	7.42	4.03	0.48	5.97	2.76	0.00	T	3.87	1.38	0.00	T	2.61	0.90	0.00	T
	50k	3.96	1.88	0.48	5.10	2.16	0.00	F	2.90	0.86	0.00	T	1.87	0.58	0.00	T
	100k	2.90	1.48	0.48	4.62	1.95	0.00	T	2.38	0.74	0.00	T	1.37	0.52	0.00	T
	150k	1.65	0.88	0.00	4.61	1.93	0.00	T	2.36	0.71	0.00	F	1.37	0.52	0.00	F
	200k	1.65	0.86	0.00	4.17	1.87	0.00	T	1.86	0.70	0.00	T	1.37	0.52	0.00	F
	500k	1.84	0.41	0.00	4.13	1.84	0.00	T	1.86	0.69	0.00	T	1.37	0.52	0.00	T
	1M	0.60	0.24	0.00	4.13	1.84	0.00	T	1.86	0.69	0.00	T	1.37	0.52	0.00	T
11	**1k**	14.79	7.08	0.99	13.22	4.23	0.00	T	24.37	5.79	0.00	T	7.30	2.05	0.00	T
	2k	9.53	4.83	0.00	8.56	2.76	0.00	T	23.01	4.80	0.00	F	6.22	1.47	0.00	T
	5k	7.18	4.07	1.77	6.09	1.67	0.00	T	20.07	3.77	0.00	F	2.71	1.07	0.00	T
	10k	5.64	2.13	0.00	5.02	1.37	0.00	F	19.67	3.49	0.00	T	2.12	0.91	0.00	T
	50k	1.81	0.64	0.12	3.39	0.95	0.00	T	19.32	3.29	0.00	T	1.84	0.82	0.00	F
	100k	1.70	0.66	0.00	3.03	0.89	0.00	T	19.20	3.28	0.00	T	1.77	0.81	0.00	T
	150k	1.02	0.32	0.00	2.97	0.88	0.00	T	19.20	3.28	0.00	T	1.65	0.81	0.00	T
	200k	1.30	0.42	0.00	3.03	0.87	0.00	T	19.20	3.27	0.00	T	1.65	0.81	0.00	T
	500k	0.26	0.09	0.00	2.85	0.86	0.00	T	19.20	3.27	0.00	T	1.65	0.81	0.00	T
	1M	0.40	0.06	0.00	2.85	0.86	0.00	T	19.20	3.27	0.00	T	1.65	0.81	0.00	T

Continued on next page

Table C.7: Summary of results of reduced search SA compared with full basic SA

		Basic SA			**SAM20**				**SAM50**				**SAM100**			
Dataset	**N**	***Max* %**	***Ave* %**	***Min* %**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**	***Max* %**	***Ave* %**	***Min* %**	**$\|t\| > t_{crit}$**
12	**1k**	14.71	9.25	4.29	16.15	2.02	0.13	T	11.06	2.74	0.05	T	8.99	1.52	0.13	T
	2k	9.83	5.23	0.97	13.78	1.69	0.00	T	8.99	2.09	0.05	T	7.77	1.13	0.00	T
	5k	8.56	3.56	1.41	11.47	1.19	0.00	F	6.96	1.61	0.05	F	6.05	0.84	0.00	T
	10k	5.51	2.76	0.22	11.30	0.82	0.00	F	6.46	1.45	0.05	F	5.79	0.72	0.00	T
	50k	3.60	1.15	0.13	9.62	0.34	0.00	T	5.38	1.25	0.00	T	5.31	0.61	0.00	T
	100k	2.32	0.95	0.08	9.36	0.27	0.00	T	5.13	1.20	0.00	T	5.08	0.60	0.00	T
	150k	1.65	0.49	0.00	9.13	0.25	0.00	T	5.31	1.19	0.00	T	5.08	0.59	0.00	T
	200k	1.02	0.45	0.13	9.31	0.25	0.00	T	5.13	1.19	0.00	T	5.08	0.59	0.00	T
	500k	0.47	0.22	0.00	8.84	0.22	0.00	T	5.08	1.19	0.00	T	5.08	0.59	0.00	T
	1M	0.23	0.08	0.00	8.79	0.22	0.00	T	5.08	1.19	0.00	T	5.08	0.59	0.00	T
Avg	**1k**	14.83	7.37	2.04	11.64	6.19	3.41	-	12.08	5.82	3.32	-	6.07	1.83	0.01	-
	2k	11.23	5.38	1.01	10.41	5.48	3.30	-	9.68	5.20	3.48	-	4.64	1.34	0.02	-
	5k	7.43	3.72	1.04	8.13	4.82	3.26	-	8.10	4.59	3.21	-	3.28	0.91	0.00	-
	10k	5.78	2.73	0.43	7.38	4.48	3.23	-	7.49	4.36	3.21	-	2.56	0.71	0.00	-
	50k	2.71	1.24	0.12	6.22	4.03	3.20	-	6.57	4.06	3.20	-	1.94	0.52	0.00	-
	100k	1.91	0.87	0.09	5.85	3.94	3.20	-	6.41	4.01	3.20	-	1.80	0.50	0.00	-
	150k	1.56	0.64	0.03	5.80	3.91	3.20	-	6.37	4.00	3.20	-	1.76	0.50	0.00	-
	200k	1.29	0.53	0.01	5.69	3.89	3.20	-	6.33	3.99	3.20	-	1.74	0.50	0.00	-
	500k	0.86	0.27	0.00	5.58	3.86	3.20	-	6.28	3.99	3.20	-	1.74	0.50	0.00	-
	1M	0.51	0.15	0.00	5.50	3.85	3.20	-	6.28	3.99	3.20	-	1.74	0.50	0.00	-

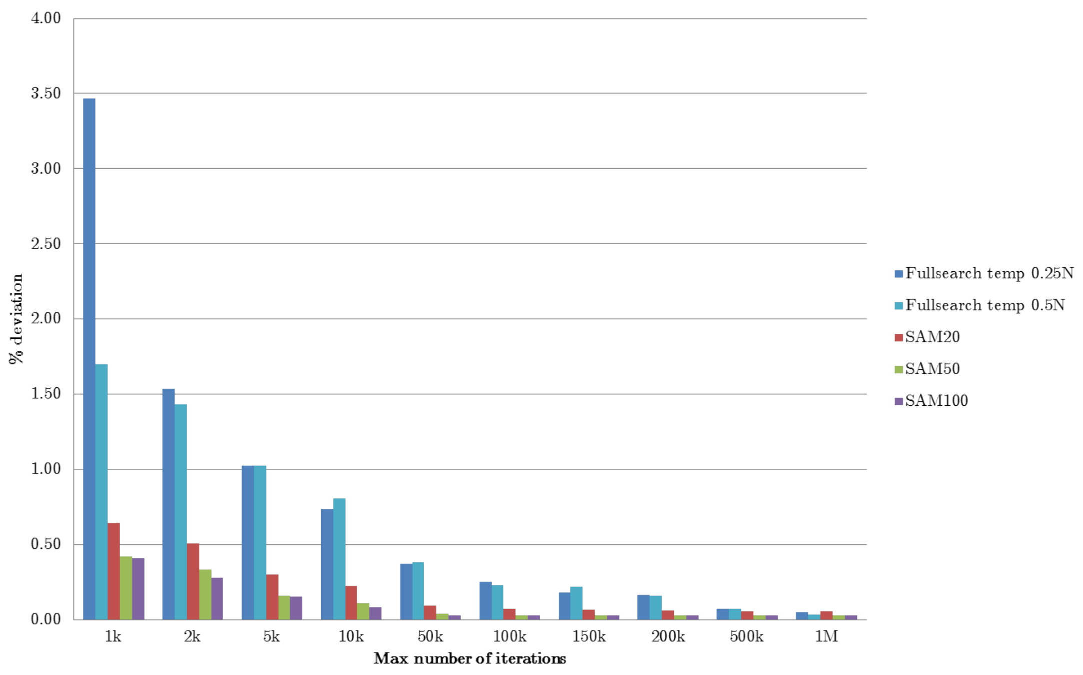


Figure C.4: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 0

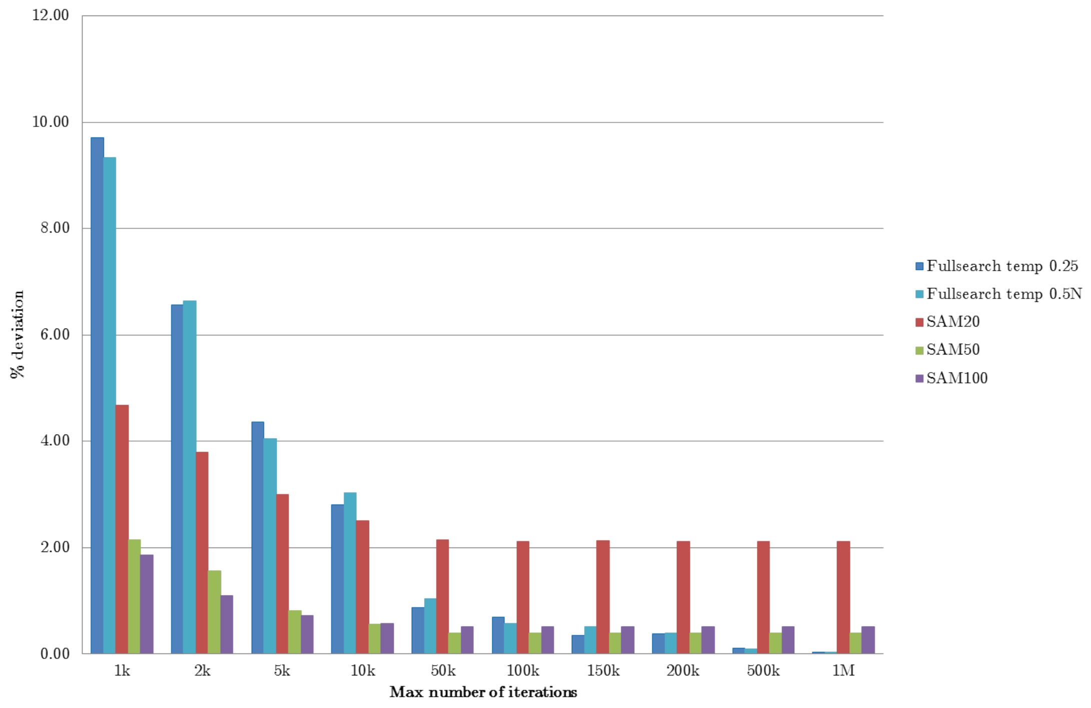


FIGURE C.5: Comparison of percentage deviation of results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 1

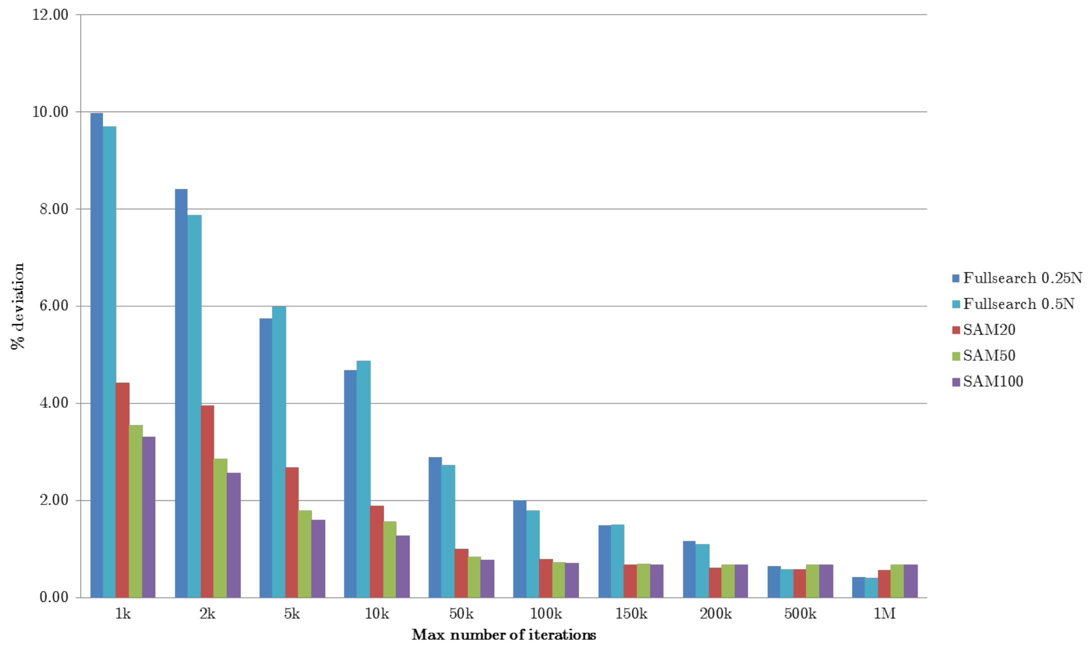


FIGURE C.6: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 2

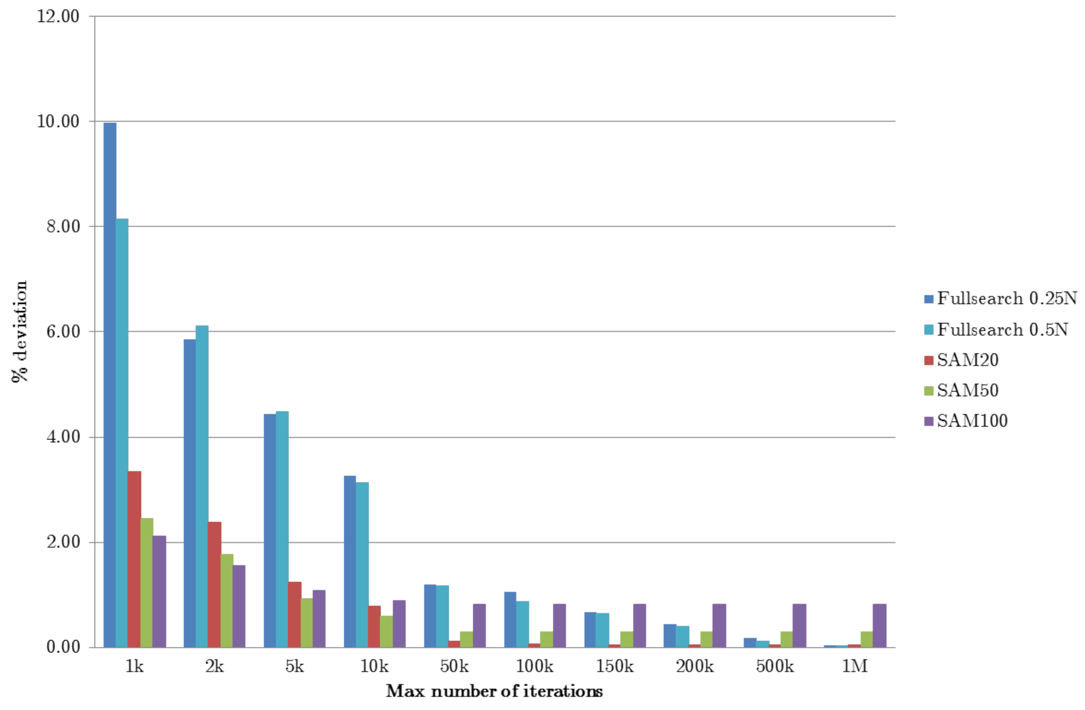


FIGURE C.7: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 3

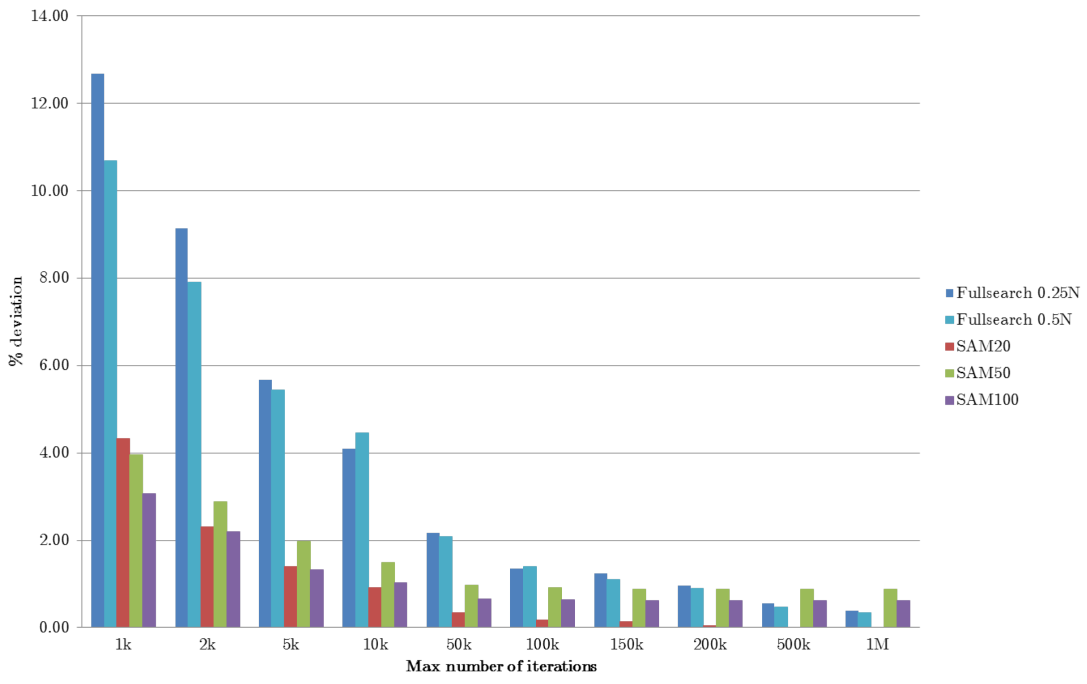


FIGURE C.8: Comparison of percentage deviation of results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 4

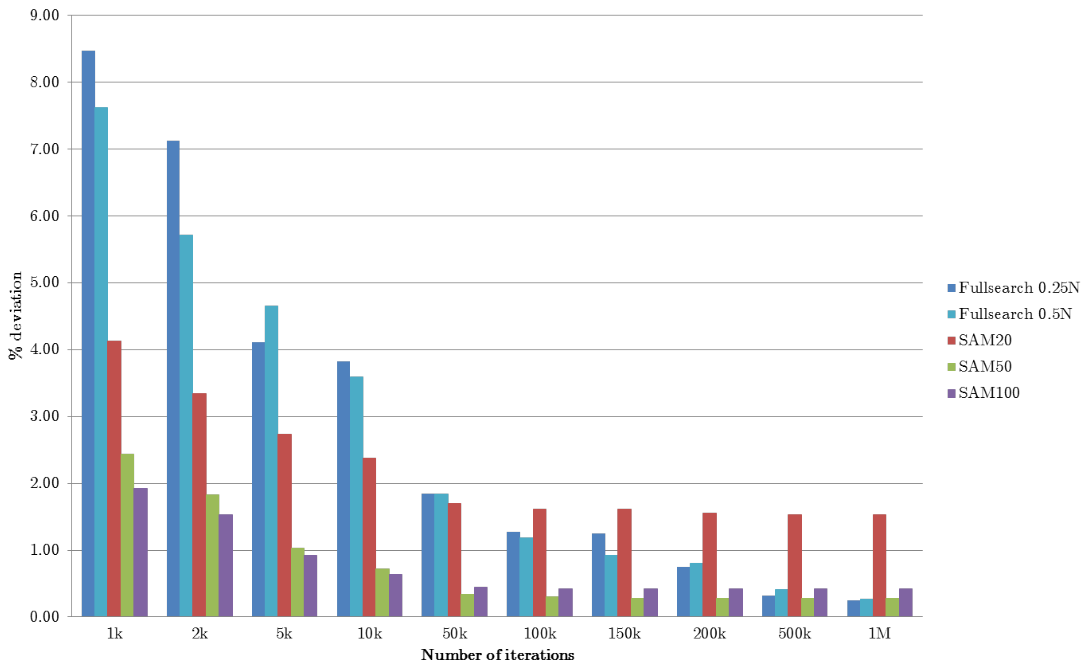


FIGURE C.9: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 5

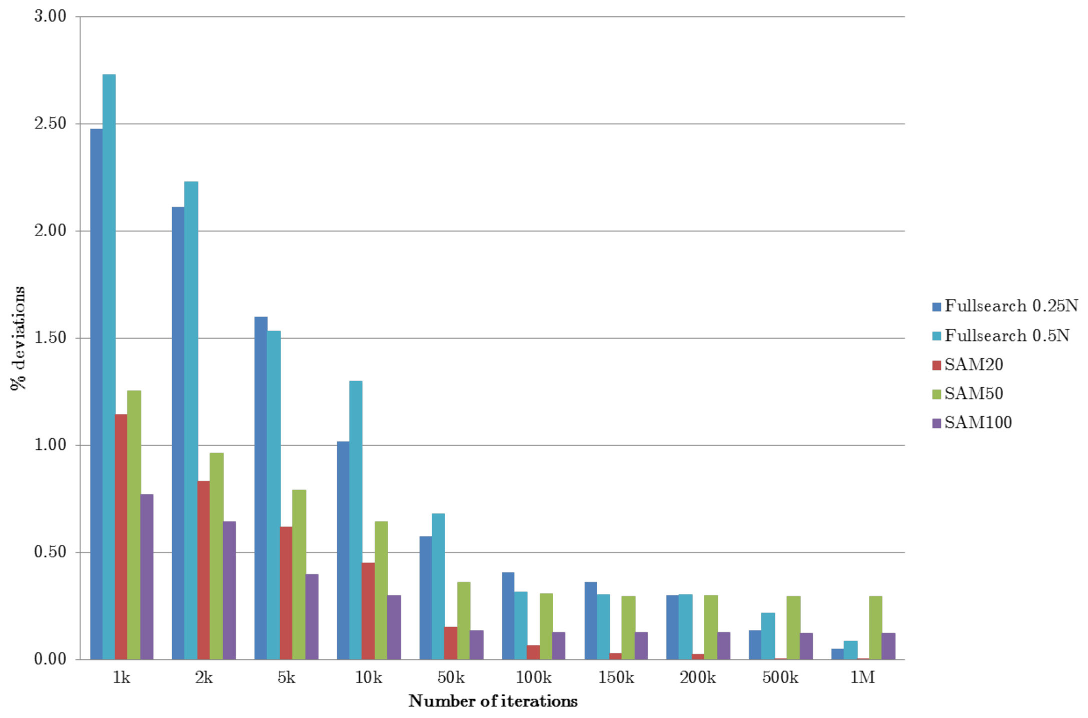


FIGURE C.10: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 6

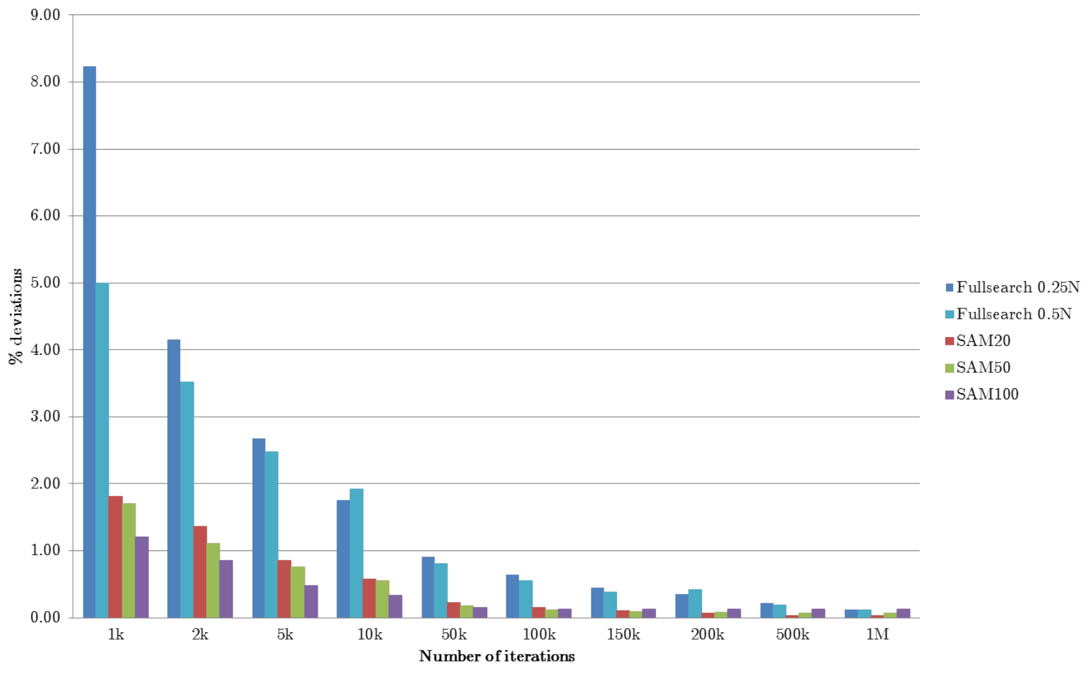


FIGURE C.11: Comparison of percentage deviation of results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on x-axis: dataset 7

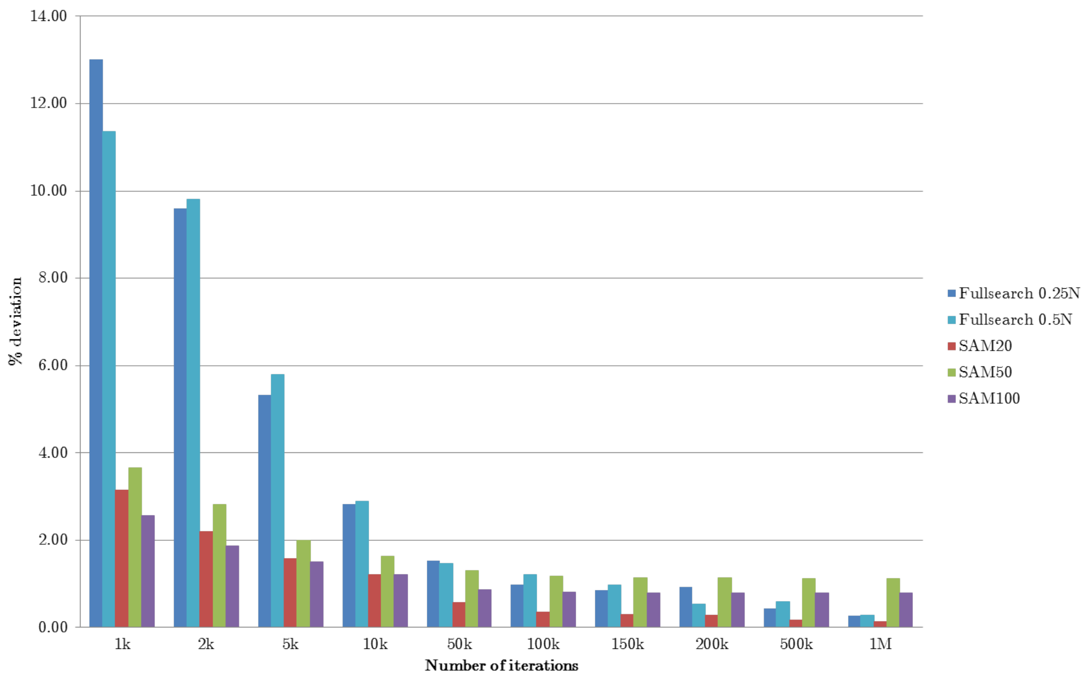


FIGURE C.12: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 8

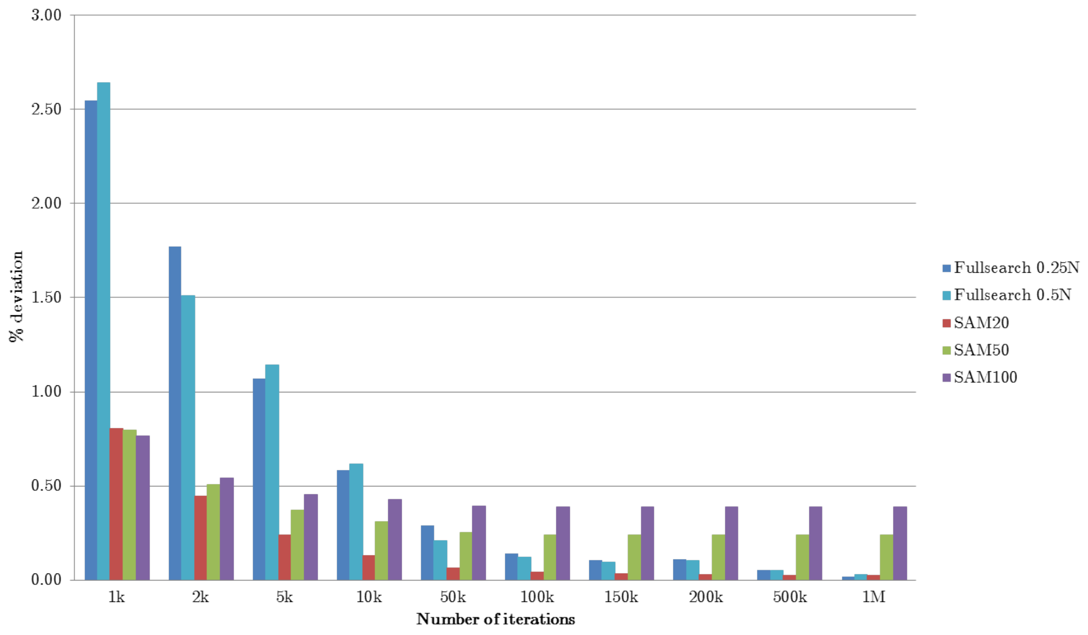


Figure C.13: Comparison of percentage deviation of results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on x-axis: dataset 9

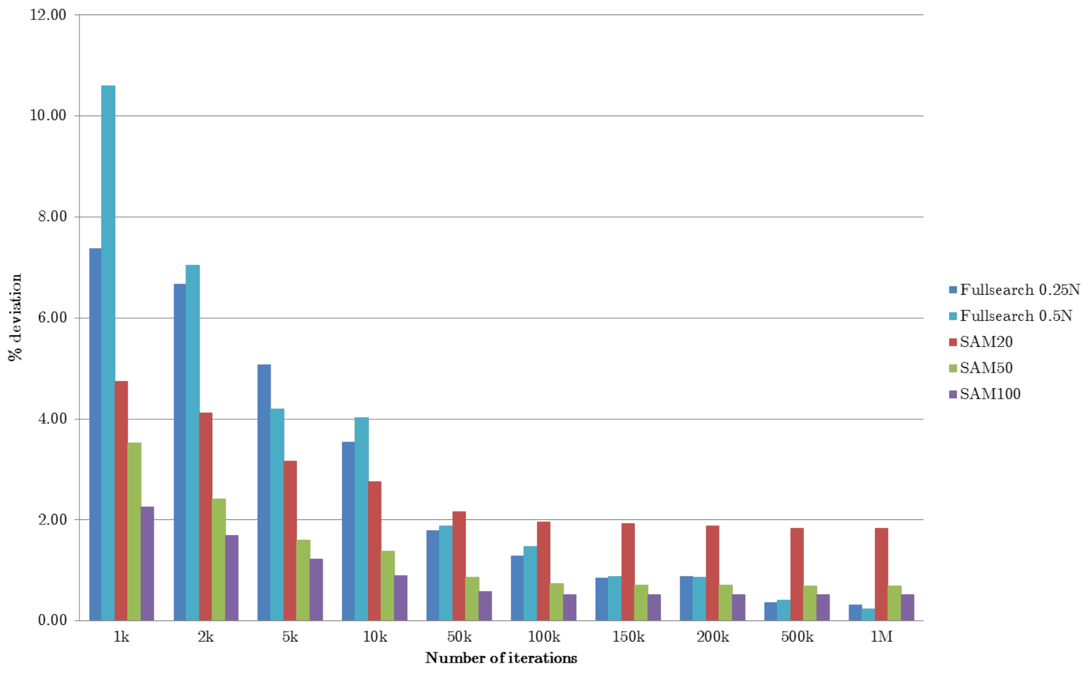


Figure C.14: Comparison of percentage deviation of results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on x-axis: dataset 10

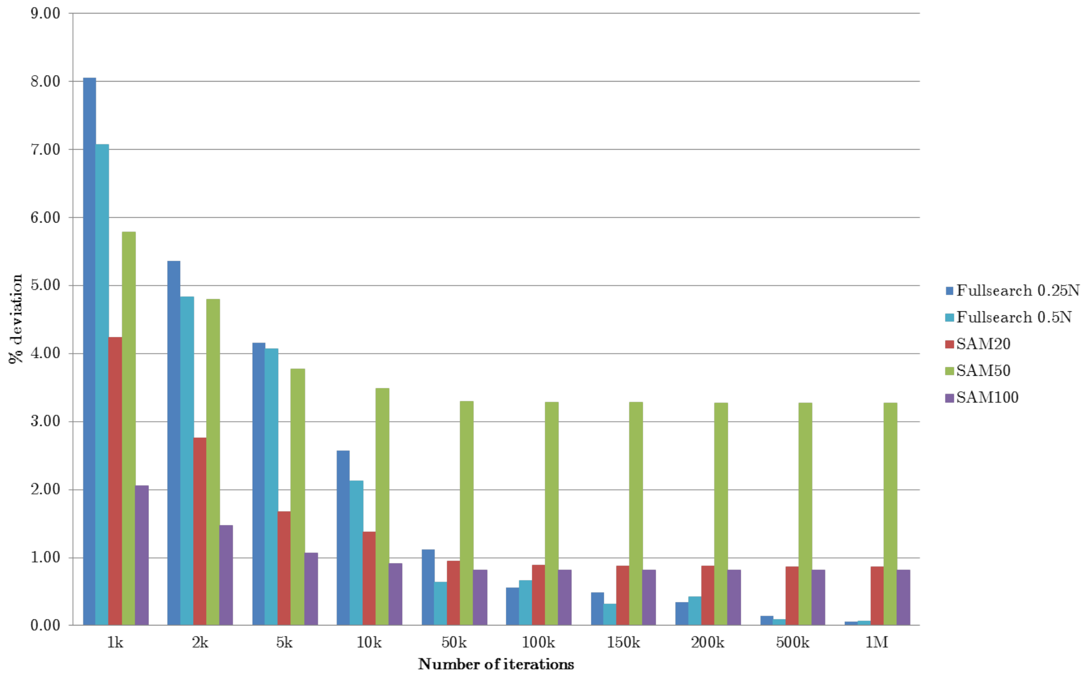


Figure C.15: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 11

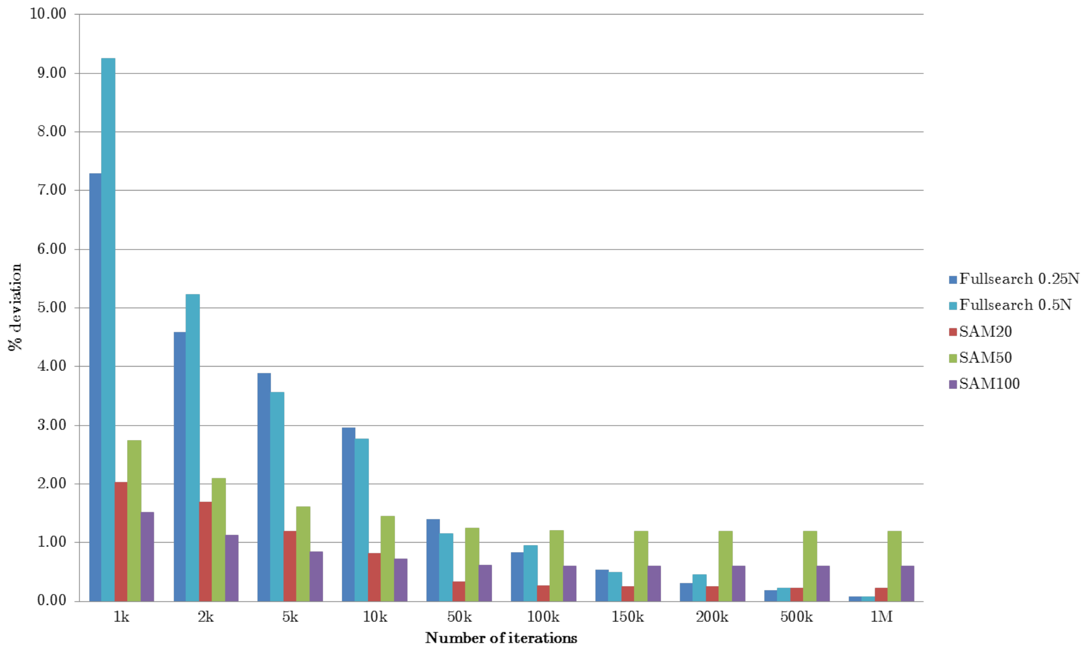


Figure C.16: Comparison of percentage deviation of the results from basic SA with two temperatures, SAM20, SAM50 and SAM100 for run lengths indicated by the iteration counts on the x-axis: dataset 12

References

[1] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(C):287–326, 1979. ISSN 01675060. doi: 10.1016/S0167-5060(08)70356-X.

[2] M.R. Garey and D.S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and company, New York, 1979. ISBN 0716710447.

[3] N.G. Hall and M.E. Posner. Generating Experimental Data for Computational Testing with Machine Scheduling Applications. *Operations Research*, 49(6):854–865, 2001.

[4] A. Janiak, W.A. Janiak, T. Krysiak, and T. Kwiatkowski. A survey on scheduling problems with due windows. *European Journal of Operational Research*, 242:347–357, 2015. doi: 10.1016/j.ejor.2014.09.043.

[5] Michael L. Pinedo. *Scheduling Theory, Algorithms, and Systems.* Springer Science+Business Media, LLC, New York, USA, third edition, 2008. ISBN 9780387789347.

[6] A.J. Page, T.M. Keane, and T.J. Naughton. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *Journal of Parallel and Distributed Computing*, 70:758–766, 2010. ISSN 07437315. doi: 10.1016/j.jpdc.2010.03.011.

[7] Thomas Hanne and Stefan Nickel. A Multi-Objective Evolutionary Algorithm for Scheduling and Inspection Planning in Software Development Projects. *Berichte des Fraunhofer ITWM*, 42(2003), 2003.

[8] J.T. Saraiva, M.L Pereira, V.T. Mendes, and J.C. Sousa. A Simulated Annealing based approach to solve the generator maintenance scheduling problem. *Electric Power Systems Research*, 81:1283–1291, 2011. doi: 10.1016/j.epsr.2011.01.013. URL www.elsevier.com/locate/epsr.

[9] B. Han, W. Zhang, X. Lu, and Y. Lin. Parallel-machine configurations with a single customer: Minimizing the makespan and delivery cost. *European Journal of Operational Research*, 000:1–11, 2015. ISSN 0377-2217. doi: 10.1016/j.ejor.2015.02.008. URL http://dx.doi.org/10.1016/j.ejor.2015.02.008.

[10] A. Khattab and M.A. Bayoumi. Standardization of cognitive radio networking: a comprehensive survey. *Annales des Telecommunications/Annals of Telecommunications*, 70(11-12):465–477, 2015. ISSN 19589395. doi: 10.1007/s12243-015-0468-5.

[11] Haris Kremo and Onur Altintas. On detecting spectrum opportunities for cognitive vehicular networks in the TV white space. *Journal of Signal Processing Systems*, 73(3):243–254, 2013. ISSN 19398018. doi: 10.1007/s11265-013-0765-z.

[12] IEEE Computer Society. *IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Television White Spaces Operation.* New York, USA, 2013. ISBN 9780738187488. doi: 10.1109/IEEESTD.2009.5307322. URL http://ieeexplore.ieee.org/servlet/opac?punumber=2408.

[13] IEEE. Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements Part 19: TV White Space Coexistence Methods, 2014.

[14] S. Filin, K. Ishizu, and H. Harada. IEEE draft standard P1900.4a for architecture and interfaces for dynamic spectrum access networks in white space frequency bands: Technical overview and feasibility study. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, pages 15–20, 2010. doi: 10.1109/PIMRCW.2010.5670353.

[15] IEEE. Draft Standard for Wireless Regional Area Networks Part 22 : Cognitive Wireless RAN Medium Access Control (MAC) and Physical Layer (PHY) specifications : Policies and procedures for operation in the TV Bands. 2010.

[16] A. Hamidinia, S. Khakabimamaghani, M.M. Mazdeh, and M. Jafari. A genetic algorithm for minimizing total tardiness/earliness of weighted jobs in a batched delivery system. *Computers and Industrial Engineering*, 62(1):29–38, 2012. ISSN 03608352. doi: 10.1016/j.cie.2011.08.014. URL http://dx.doi.org/10.1016/j.cie.2011.08.014.

[17] F. Jolai, M. Rabbani, S. Amalnick, A. Dabaghi, M. Dehghan, and M.Y. Parast. Genetic algorithm for bi-criteria single machine scheduling problem of minimizing

maximum earliness and number of tardy jobs. *Applied Mathematics and Computation*, 194:552–560, 2007. ISSN 00963003. doi: 10.1016/j.amc.2007.04.063.

[18] M.C. Gupta, Y.P. Gupta, and A. Kumar. Minimizing flow time variance in a single machine system using genetic algorithms. *European Journal of Operational Research*, 70:289–303, 1993. ISSN 03772217. doi: 10.1016/0377-2217(93)90240-N.

[19] A.P. Adewole, K. Otubamowo, and T.O. Egunjobi. A Comparative Study of Simulated Annealing and Genetic Algorithm for Solving the Travelling Salesman Problem. *International Journal of Applied Information Systems (IJAIS)*, 4(4): 6–12, 2012.

[20] T.W. Manikas and J.T. Cain. Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem. Technical report, The University of Pittsburgh, Pittsburgh, PA, 1996.

[21] Tian Jungai and Xu Hongjun. Optimizing Arrival Flight Delay Scheduling Based on Simulated Annealing Algorithm. *Physics Procedia*, 33:348–353, 2012. ISSN 1875-3892. doi: 10.1016/j.phpro.2012.05.073. URL http://dx.doi.org/10.1016/j.phpro.2012.05.073.

[22] W.H.M. Raaymakers and J.A. Hoogeveen. Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, 126:131–151, 2000. ISSN 03772217. doi: 10.1016/S0377-2217(99)00285-4.

[23] W.A. Massey, K.G. Ramakrishnan, M. Aravamudan, and G. Pai. Scheduling Algorithms for Downlink Services in Wireless Networks : A Markov Decision Process Approach. *Communications Society*, pages 4038–4042, 2004. doi: 10.1109/GLOCOM.2004.1379125.

[24] H.A.J. Crauwels, C.N. Potts, and L.N. Van Wassenhove. Local search heuristics for single-machine scheduling with batching to minimize the number of late jobs. *European Journal of Operational Research*, 90(1996):200–213, apr 1996. ISSN 03772217. doi: 10.1016/0377-2217(95)00349-5. URL http://www.sciencedirect.com/science/article/pii/0377221795003495.

[25] T.C.Edwin; Cheng, Valery S. Gordon, and Mikhail Y. Kovalyov. Single Machine Scheduling with Batch Deliveries. *European Journal of Operational Research*, 94 (2):277–283, oct 1996. ISSN 03772217. doi: 10.1016/0377-2217(96)00127-0. URL http://www.sciencedirect.com/science/article/pii/0377221796001270.

[26] Martin Feldmann and Dirk Biskup. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers and Industrial Engineering*, 44:307–323, 2003. ISSN 03608352. doi: 10.1016/S0360-8352(02)00181-X.

[27] Candace Arai Yano and Yeong-dae Kim. Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal of Operational Research*, 52:167–178, 1991.

[28] J. Bank and F. Werner. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling*, 33:363–383, 2001. ISSN 08957177. doi: 10.1016/S0895-7177(00)00250-8.

[29] V. Sridharan and Z. Zhou. A decision theory based scheduling procedure for single-machine weighted earliness and tardiness problems. *European Journal of Operational Research*, 94(2):292–301, 1996. ISSN 03772217. doi: 10.1016/0377-2217(96)00133-6.

[30] G. Wan and B.P.C. Yen. Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research*, 142:271–281, 2002. ISSN 03772217. doi: 10.1016/S0377-2217(01)00302-2.

[31] W.K. Yeung, C. Oguz, and T.C.E. Cheng. Single-machine scheduling with a common due window. *Computers and Operations Research*, 28:157–175, 2001. ISSN 03050548. doi: 10.1016/S0305-0548(99)00097-0.

[32] B. Esteve, C. Aubijoux, A. Chartier, and V. T'kindt. A recovering beam search algorithm for the single machine Just-in-Time scheduling problem. *European Journal of Operational Research*, 172(3):798–813, aug 2006. ISSN 03772217. doi: 10.1016/j.ejor.2004.11.014. URL http://www.sciencedirect.com/science/article/pii/S0377221704008574.

[33] C.M. Hino, D.P. Ronconi, and A.B. Mendes. Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. *European Journal of Operational Research*, 160:190–201, 2005. ISSN 03772217. doi: 10.1016/j.ejor.2004.03.006.

[34] Han Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167:592–623, 2005. ISSN 03772217. doi: 10.1016/j.ejor.2004.07.011.

[35] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1):59–66, 1956.

[36] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnoy Kan, and David B. Shmoys. Minmax criteria.

[37] Natallia Kokash. An introduction to heuristic algorithms. 2005.

[38] C.M. Bishop and N.M. Nasrabadi. *Pattern Recognition and Machine Learning*, volume 4. Springer Science+Business Media, LLC, Singapore, 2006. ISBN 9780387310732. doi: 10.1117/1.2819119. URL http://www.library.wisc.edu/selectedtocs/bg0137.pdf.

[39] Julian Besag. Markov Chain Monte Carlo for Statistical Inference. 2001.

[40] J.C. Spall. Estimation via Markov chain Monte Carlo. *IEEE Control Systems Magazine*, 23(April):34–45, 2003. ISSN 0272-1708. doi: 10.1109/MCS.2003.1188770. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1188770.

[41] C.J. Geyer. Introduction to Markov Chain Monte Carlo. *Handbook of Markov Chain Monte Carlo*, (1990):3–48, 2002. URL http://www.mcmchandbook.net/index.html.

[42] Gareth O. Roberts and Jeffrey S. Rosenthal. General state space Markov chains and MCMC algorithms. *Probability Surveys*, 1:20–71, 2004. ISSN 1549-5787. doi: 10.1214/154957804100000024. URL http://www.i-journals.org/ps/viewarticle.php?id=15{&}layout=abstract.

[43] Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18(4):343–373, 2008. ISSN 09603174. doi: 10.1007/s11222-008-9110-y.

[44] S.P. Brooks. Markov Chain Monte Carlo Method and its Application. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 47(1):69–100, 2010.

[45] D.J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*, volume 22. 2003. ISBN 0521642981. doi: 10.1017/S026357470426043X. URL http://www.journals.cambridge.org/abstract{_}S026357470426043X.

[46] M.K. Cowles and B.P. Carlin. Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review. *Journal of the American Statistical Association*, 91 (434):883–904, 1996.

[47] S.P Brooks and G.O. Roberts. Assessing Convergence of Markov Chain Monte Carlo Algorithms. *Statistics and Computing*, 8:319–335, 1997. ISSN 0717-6163. doi: 10.1007/s13398-014-0173-7.2.

[48] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science, New Series*, 220(4598):671–680, 1983.

[49] K.C. Tan and R. Narasimhan. Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. *Omega, International Journal of Management Science*, 25(6):619–634, 1997. ISSN 03050483. doi: 10.1016/S0305-0483(97)00024-8.

[50] J. Jozefowska, M. Mika, R. Rozycki, G. Waligora, and J. Weglarz. Simulated Annealing for Multi-Mode Resource-Constrained Project Scheduling. *Annals of Operations Research*, 102:137–155, 2001.

[51] Dong-Won Kim, Kyong-Hee Kim, Wooseung Jang, and F Frank Chen. Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18:223–231, 2002. ISSN 07365845. doi: 10.1016/S0736-5845(02)00013-3.

[52] I. Mahdavi, V. Kayvanfar, and G.M. Komaki. Minimizing total tardiness and earliness problem with controllable processing times using an effective heuristic. In *40th International Conference on Computers and Industrial Engineering: Soft Computing Techniques for Advanced Manufacturing and Service Systems, CIE40*, 2010. ISBN 9781424472956. doi: 10.1109/ICCIE.2010.5668166.

[53] T. Loukil, J. Teghem, and D. Tuyttens. Solving multi-objective production scheduling problems using metaheuristics. *European Journal of Operational Research*, 161:42–61, 2005. ISSN 03772217. doi: 10.1016/j.ejor.2003.08.029.

[54] C.C. Wu, H.M. Chen, S.R. Cheng, C.J. Hsu, and W.H. Wu. Simulated annealing approach for the single-machine total late work scheduling problem with a position-based learning. In *IEEE 18th International Conference on Industrial Engineering and Engineering Management, IE and EM*, pages 839–843, 2011. ISBN 9781612844473. doi: 10.1109/IEEM.2011.6035289.

[55] Wen-hung Wu. Solving a two-agent single-machine learning scheduling problem. *International Journal of Computer Integrated Manufacturing*, 27(1):20–35, 2014.

[56] M. Wang, C. Feng, and T. Zhang. A simulated annealing algorithm for the transmission mode selection of multicast services in LTE networks. In *The 14th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–5, 2011. ISBN 978-2-908849-26-4.

[57] S. Fichera, F. Cappadonna, A. Costa, and A. Fichera. A Simulated Annealing Algorithm for Single Machine Scheduling Problem with Release Dates, Learning

and Deteriorating Effects. In *Proceedings of the World Congress on Engineering*, volume I, pages 6–8, London, UK, 2013. ISBN 9789881925107. URL http://www.iaeng.org/publication/WCE2013/WCE2013{_}pp584-586.pdf.

[58] C.N. Potts and L.N . Van Wassenhove. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research*, 33(2):363–377, 1985.

[59] G. Wan and B.P.C. Yen. Single machine scheduling to minimize total weighted earliness subject to minimal number of tardy jobs. *European Journal of Operational Research*, 195(1):89–97, may 2009. ISSN 03772217. doi: 10.1016/j.ejor.2008.01.029. URL http://www.sciencedirect.com/science/article/pii/S0377221708001628http://dx.doi.org/10.1016/j.ejor.2008.01.029.

[60] M.M. Mazdeh, M. Sarhadi, and K.S. Hindi. A branch-and-bound algorithm for single-machine scheduling with batch delivery minimizing flow times and delivery costs. *European Journal of Operational Research*, 183:74–86, 2007. ISSN 03772217. doi: 10.1016/j.ejor.2006.09.087.

[61] Mitchell Melanie. *An introduction to genetic algorithms.* MIT Press, Cambridge, Massachusetts, fifth edition, 1996. ISBN 0-262-13316-4. doi: 10.1016/S0898-1221(96)90227-8.

[62] A.E. Zade and M.B. Fakhrzad. A Dynamic Genetic Algorithm for Solving a Single Machine Scheduling Problem with Periodic Maintenance. *ISRN Industrial Engineering*, 2013, 2013.

[63] Murat Köksalan and Ahmet Burak Keha. Using genetic algorithms for single-machine bicriteria scheduling problems. *European Journal of Operational Research*, 145(3):543–556, 2003. ISSN 03772217. doi: 10.1016/S0377-2217(02)00220-5. URL http://www.sciencedirect.com/science/article/pii/S0377221702002205.

[64] Mieczysław Wodecki. A block approach to earliness-tardiness scheduling problems. *International Journal of Advanced Manufacturing Technology*, 40:797–807, 2009. ISSN 02683768. doi: 10.1007/s00170-008-1395-7.

[65] Antoine Jouglet, David Savourey, Jacques Carlier, and Philippe Baptiste. Dominance-based heuristics for one-machine total cost scheduling problems. *European Journal of Operational Research*, 184(3):879–899, feb 2008. ISSN 03772217. doi: 10.1016/j.ejor.2006.11.036. URL http://www.sciencedirect.com/science/article/pii/S0377221706012367.

[66] F.S. Erenay, I. Sabuncuoglu, A. Toptal, and M.K. Tiwari. New solution methods for single machine bicriteria scheduling problem: Minimization of average flowtime

and number of tardy jobs. *European Journal of Operational Research*, 201:89–98, 2010. ISSN 03772217. doi: 10.1016/j.ejor.2009.02.014.

[67] D. Danneberg, T. Tautenhahn, and F. Werner. A comparison of heuristic algorithms for flow shop scheduling problems with setup times and limited batch size. *Mathematical and Computer Modelling*, 29(99):101–126, 1999. ISSN 08957177. doi: 10.1016/S0895-7177(99)00085-0.

[68] Candace A. Yano and Y. D. Kim. Algorithms for single machine scheduling problems minimizing tardiness and earliness. Technical report, 1986.

[69] D.P. Ronconi and M.S. Kawamura. The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm. *Computational & applied mathematics*, 29(2):107–124, 2010. ISSN 01018205. doi: 10.1590/S1807-03022010000200002.

[70] CA. Yano and Y. Kim. Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal of Operational Research*, 52(2):167–178, 1991. ISSN 03772217. doi: 10.1016/0377-2217(91)90078-A. URL http://www.sciencedirect.com/science/article/pii/037722179190078A.

[71] K. Tan, R. Narasimhan, P.A. Rubin, and G.L. Ragatz. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. *Omega, International Journal of Management Science*, 28:313–326, 2000. ISSN 03050483. doi: 10.1016/S0305-0483(99)00050-X.

[72] Suresh Chand, Hans Schneeberger, and West Lafayette. Theory and Methodology Single machine scheduling to minimize weighted earliness subject to no tardy jobs. *European Journal of Operational Research*, 34:221–230, 1988.

[73] Wooseung Jang. Dynamic scheduling of stochastic jobs on a single machine. *European Journal of Operational Research*, 138:518–530, 2002. ISSN 03772217. doi: 10.1016/S0377-2217(01)00174-6.

[74] Toshihide Ibaraki and Yuichi Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76:72–82, 1994. ISSN 03772217. doi: 10.1016/0377-2217(94)90007-8.

[75] V.S. Gordon and V.A. Strusevich. Single machine scheduling and due date assignment with positionally dependent processing times. *European Journal of Operational Research*, 198(1):57–62, 2009. ISSN 03772217. doi: 10.1016/j.ejor.2008.07.044. URL http://dx.doi.org/10.1016/j.ejor.2008.07.044.

[76] M. Ji and T.C.E. Cheng. Batch scheduling of simple linear deteriorating jobs on a single machine to minimize makespan. *European Journal of Operational Research*, 202(1):90–98, apr 2010. ISSN 03772217. doi: 10.1016/j.ejor.2009.05.021. URL http://www.sciencedirect.com/science/article/pii/S0377221709003518.

[77] Francis Sourd. Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, 165:82–96, 2005. ISSN 03772217. doi: 10.1016/j.ejor.2004.01.025.

[78] Shunji Tanaka. An Exact Algorithm for the Single-Machine Earliness-Tardiness Scheduling Problem. *Just-in-Time Systems*, pages 21–41, 2012. doi: 10.1007/978-1-4614-1123-9.

[79] Dvir Shabtay. The single machine serial batch scheduling problem with rejection to minimize total completion time and total rejection cost. *European Journal of Operational Research*, 233(1):64–74, 2014. ISSN 03772217. doi: 10.1016/j.ejor.2013.08.013. URL http://dx.doi.org/10.1016/j.ejor.2013.08.013.

[80] S.H. Chen, M.C. Chen, P.C. Chang, and Y.M. Chen. EA/G-GA for single machine scheduling problems with earliness/tardiness costs. *Entropy*, 13(6):1152–1169, 2011. ISSN 10994300. doi: 10.3390/e13061152.

[81] Virginia Yannibelli and Analía Amandi. Expert Systems with Applications Hybridizing a multi-objective simulated annealing algorithm with a multi-objective evolutionary algorithm to solve a multi-objective project scheduling problem. *Expert Systems With Applications*, 40(7):2421–2434, 2013. ISSN 0957-4174. doi: 10.1016/j.eswa.2012.10.058. URL http://dx.doi.org/10.1016/j.eswa.2012.10.058.

[82] S.P. Ali and M. Bijari. Minimizing maximum earliness and tardiness on a single machine using a novel heuristic approach. In *Proceedings of the 2012 International Conference on Industrial Engineering and Operations Management*, pages 1091–1097, Istanbul, Turkey, 2012.

[83] M. Ji, J. Ge, K. Chen, and T.C.E. Cheng. Single-machine due-window assignment and scheduling with resource allocation, aging effect, and a deteriorating rate-modifying activity. *Computers and Industrial Engineering*, 66(4):952–961, 2013. ISSN 03608352. doi: 10.1016/j.cie.2013.08.020. URL http://dx.doi.org/10.1016/j.cie.2013.08.020.

[84] Bo Cheng and Ling Cheng. Note on "Single-machine due-window assignment and scheduling with resource allocation, aging effect, and a deteriorating rate-modifying activity". *Computers and Industrial Engineering*, 78:320–322, 2014.

ISSN 03608352. doi: 10.1016/j.cie.2013.08.020. URL http://dx.doi.org/10.1016/j.cie.2014.07.013.

[85] S.R. Gupta and J.S. Smith. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, 175(2):722–739, dec 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.05.018. URL http://www.sciencedirect.com/science/article/pii/S0377221705005060.

[86] S.G. Kolliopoulos and G. Steiner. Approximation algorithms for scheduling problems with a modified total weighted tardiness objective. *Operations Research Letters*, 35(5):685–692, sep 2007. ISSN 0167-6377. doi: http://dx.doi.org/10.1016/j.orl.2006.12.002. URL http://www.sciencedirect.com/science/article/pii/S0167637706001337.

[87] J. Talebi, H. Badri, F. Ghaderi, and E. Khosravian. An efficient scatter search algorithm for minimizing earliness and tardiness penalties in a single-machine scheduling problem with a common due date. In *IEEE Congress on Evolutionary Computation, CEC*, pages 1012–1018, 2009. ISBN 9781424429592. doi: 10.1109/CEC.2009.4983056.

[88] P.M. França, A. Mendes, and P. Moscato. A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132(1):224–242, jul 2001. ISSN 03772217. doi: 10.1016/S0377-2217(00)00140-5. URL http://www.sciencedirect.com/science/article/pii/S0377221700001405.

[89] M.F. Rego, M.J.F. Souza, and J.E.C. Arroyo. Multi-objective algorithms for the single machine scheduling problem with sequence-dependent family setups. In *Proceedings - International Conference of the Chilean Computer Science Society, SCCC*, pages 142–151, 2013. ISBN 9781479929375. doi: 10.1109/SCCC.2012.24.

[90] Z. Ruiguo and L. Jiejia. A neighborhood search algorithm for one-machine scheduling problem with time lags. *Chinese Control and Decision Conference*, pages 1937–1940, 2009. doi: 10.1109/CCDC.2009.5191609.

[91] Davide Anghinolfi and Massimo Paolucci. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 193:73–85, 2009. ISSN 03772217. doi: 10.1016/j.ejor.2007.10.044.

[92] P. Perez-Gonzalez and J.M. Framinan. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, 235(1):

1–16, 2014. ISSN 03772217. doi: 10.1016/j.ejor.2013.09.017. URL http://dx.doi.org/10.1016/j.ejor.2013.09.017.

[93] Baruch Mor and Gur Mosheiov. Single machine batch scheduling with two competing agents to minimize total flowtime. *European Journal of Operational Research*, 215(3):524–531, 2011. ISSN 03772217. doi: 10.1016/j.ejor.2011.06.037. URL http://dx.doi.org/10.1016/j.ejor.2011.06.037.

[94] L. Li, D.J. Fonseca, and D.S. Chen. Earliness-tardiness production planning for just-in-time manufacturing: A unifying approach by goal programming. *European Journal of Operational Research*, 175:508–515, 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.06.009.

[95] C.T.D. Ng, T.C.E. Cheng, and M.Y. Kovalyov. Single machine batch scheduling with jointly compressible setup and processing times. In *European Journal of Operational Research*, volume 153, pages 211–219, 2004. ISBN 0377-2217. doi: 10.1016/S0377-2217(02)00732-4.

[96] G.O. Roberts and J.S. Rosenthal. Examples of Adaptive MCMC. Technical Report 0610, University of Toronto Department of Statistics, 2008.

[97] David Luengo and Luca Martino. Fully Adaptive Gaussian Mixture Metropolis-Hastings Algorithm. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1–10, 2012. ISBN 9781479903566. URL http://arxiv.org/abs/1212.0122.

[98] Nimalan Mahendran, Z Wang, F Hamze, and N. De Freitas. Adaptive MCMC with Bayesian Optimization. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:152, 2010.

[99] H. Haario, E. Saksman, and J. Tamminen. Adaptive proposal distribution for random walk Metropolis algorithm. *Computational Statistics*, 14(3):375, 1999. ISSN 09434062. doi: 10.1007/s001800050022.

[100] A. Kramer, B. Calderhead, and N. Radde. Hamiltonian Monte Carlo methods for efficient parameter estimation in steady state dynamical systems. *BMC Bioinformatics*, 15(1):253, 2014. ISSN 1471-2105. doi: 10.1186/1471-2105-15-253. URL http://www.biomedcentral.com/1471-2105/15/253.

[101] I. Ekeland, R. Temam, J. Dean, D. Grove, C. Chambers, K.B Bruce, and E. Bertino. Hamiltonian Mechanics unter besonderer Berucksichtigung der hohreren Lehranstalten.

[102] P.J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995. ISSN 0006-3444. doi: 10.1093/biomet/82.4.711.

[103] G.O. Roberts, A. Gelman, and W.R. Gilks. Weak convergence and optimal scaling of random walk Metropolis algorithms, 1997. ISSN 10505164.

[104] M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. In *Approximation algorithms for NP-hard problems*, chapter 12, pages 482–520. 1997. ISBN 0-7803-8794-5. doi: 10.1109/GLOCOM.2004.1377963.

[105] D. Golenko-Ginzburg. Metrics in the permutation space. *Applied Mathematics Letters*, 4(2):5–7, 1991. ISSN 08939659. doi: 10.1016/0893-9659(91)90156-P.

[106] D.E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2: Generating all tuples and permutations*. Addison-Wesley Professional, 12th edition, 2009. ISBN 0321580508, 9780321580504.

[107] B.R. Heap. Permutations by interchanges. *The Computer Journal*, 6(3):293–298, 1963. ISSN 0010-4620. doi: 10.1093/comjnl/6.3.293. URL http://comjnl.oxfordjournals.org/content/6/3/293.

[108] P.P. Lukaszewicz. *Metaheuristics for Job Shop Scheduling Problem, Comparison of Effective Methods*. Master, Aarhus School of Business, 2005.

[109] Dimitris Bertsimas and John Tsitsiklis. Simulated Annealing, 1993. ISSN 0883-4237.

[110] M.B. Villarino. Ramanujan's Approximation to the nth Partial Sum of the Harmonic Series. 2004. URL http://arxiv.org/abs/math/0402354.

[111] W.E. Brown. Random Number Generation in C ++ 11. pages 1–12, 2013.

[112] C.P. Robert. The Metropolis-Hastings algorithm. 2015. URL http://arxiv.org/abs/1504.01896.

[113] P.C. Chang. A Branch and Bound Approach for Single Machine Scheduling with Earliness and Tardiness Penalties. *Computers and Mathematics with Applications*, 37(99):133–144, 1999. ISSN 08981221. doi: 10.1016/S0898-1221(99)00130-3.

[114] George Li. Single machine earliness and tardiness scheduling. *European Journal of Operational Research*, 96(3):546–558, feb 1997. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/S0377-2217(96)00062-8. URL http://www.sciencedirect.com/science/article/pii/S0377221796000628.

[115] H. Dai, X. Wu, L. Xu, and G. Chen. Practical scheduling for stochastic event capture in wireless rechargeable sensor networks. *IEEE Wireless Communications and Networking Conference, WCNC*, pages 986–991, 2013. ISSN 15253511. doi: 10.1109/WCNC.2013.6554698.

[116] E. Molaee, G. Moslehi, and M. Reisi. Minimizing maximum earliness and number of tardy jobs in the single machine scheduling problem. *Computers and Mathematics with Applications*, 60(11):2909–2919, 2010. ISSN 08981221. doi: 10.1016/j.camwa.2010.09.046. URL http://dx.doi.org/10.1016/j.camwa.2010.09.046http://linkinghub.elsevier.com/retrieve/pii/S0898122110007431.

[117] OR Library. URL ORLibrary:http://sprocket.ict.pwr.wroc.pl/?wbo/benchmarks.htm21.

[118] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, XXVII(3), 1948.

[119] A. Grilo, M.M. Macedo, and M.S. Nunes. IP QoS support in IEEE 802.11b WLANs. *Computer Communications*, 26(17):1918–1930, 2003. ISSN 01403664. doi: 10.1016/S0140-3664(03)00157-9.

[120] Dirk Biskup and Martin Feldmann. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operations Research*, 28(8):787–801, 2001. ISSN 03050548. doi: 10.1016/S0305-0548(00)00008-3.

[121] Rym M'Hallah. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. *Computers and Operations Research*, 34:3126–3142, 2007. ISSN 03050548. doi: 10.1016/j.cor.2005.11.021.

[122] J.P. De C. M. Nogueira, J.E.C. Arroyo, H.M.M Villadiego, and L.B. Goncalves. Hybrid GRASP heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties. *Electronic Notes in Theoretical Computer Science*, 302:53–72, 2014. ISSN 15710661. doi: 10.1016/j.entcs.2014.01.020.

[123] H. Tamaki, H. Murao, and S. Kitamura. A heuristic-based hybrid solution for parallel machine scheduling problems with earliness and tardiness penalties. In *IEEE Conference on Emerging Technologies and Factory Automation. Proceedings*, volume 2, 2003. ISBN 0-7803-7937-3. doi: 10.1109/ETFA.2003.1248706.

[124] Ewa Figielska. A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources. *Computers and Industrial Engineering*,

56(1):142–151, 2009. ISSN 03608352. doi: 10.1016/j.cie.2008.04.008. URL http://dx.doi.org/10.1016/j.cie.2008.04.008.

[125] Q. Li, L. Liang, and W. Qiao. A Hybrid Robust Scheduling for Single Machine Subject to Random Machine Breakdown. In *Fourth International Workshop on Advanced Computational Intelligence*, pages 700–705, Wuhan, Hubei, China, 2011. IEEE. ISBN 9781612843759.

[126] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Number January. 2007. ISBN 0521424267. doi: 10.1088/1742-6596/1/1/035. URL http://retarget.googlecode.com/svn-history/r160/trunk/temp/Complexity/book.pdf$\delimiter"026E30F$nhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.7207{&}rep=rep1{&}type=pdf.

[127] T.H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, third edition, 2009. ISBN 9780262033848.

[128] A.M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Alan Turing: His Work and Impact*, 42(1):230–265, 1936. ISSN 00664138. doi: 10.1016/B978-0-12-386980-7.50002-2.