Rollout Algorithms for Job-Shop Scheduling

Einar Geirsson

Faculty of Industrial Engineering
University of Iceland
2012

ROLLOUT ALGORITHMS FOR JOB-SHOP SCHEDULING

Einar Geirsson

30 ECTS thesis submitted in partial fulfillment of a
Magister Scientiarum degree in Industrial engineering

Advisors
Tómas Philip Rúnarsson
Steinn Guðmundsson

Faculty Representative
Sven Þórarinn Sigurðsson

Faculty of Industrial Engineering
School of Engineering and Natural Sciences
University of Iceland
Reykjavik, May 2012

Rollout Algorithms for Job-Shop Scheduling

30 ECTS thesis submitted in partial fulfillment of a M.Sc. degree in Industrial engineering

Copyright © 2012 Einar Geirsson
All rights reserved

Faculty of Industrial Engineering
School of Engineering and Natural Sciences
University of Iceland
Hjarðarhagi 2-6
107, Reykjavik, Reykjavik
Iceland

Telephone: 525 4000

Bibliographic information:
Einar Geirsson, 2012, Rollout Algorithms for Job-Shop Scheduling, M.Sc. thesis, Faculty of Industrial Engineering, University of Iceland.

Printing: Háskólaprent, Fálkagata 2, 107 Reykjavík
Reykjavik, Iceland, May 2012

Abstract

The topic of this thesis are new approximation methods for job-shop scheduling that dispatch jobs based on statistics collected from multiple Monte Carlo rollouts. The methods use a look-ahead feature to evaluate all the jobs available for dispatching, by generating multiple feasible schedules. Previous work on rollout algorithms for combinatorial optimization problems has focused on sequentially consistent heuristics, that only search small areas of the solution space. The new algorithms widen the search space by basing the search on sequentially inconsistent algorithms. Four new algorithms are proposed: the fortified rollout algorithm, average rollout algorithm, hybrid rollout algorithm and quantile rollout algorithm. These methods differ in the way jobs are dispatched after completion of rollouts. The methods are tested on 600 job-shop problems of three different dimensions. All of the algorithms were able to generate schedules of higher quality than previous algorithms employing rollouts and the quantile rollout algorithm produced the best schedules overall, especially for larger problem instances.

Ágrip

Útspilunarreiknirit fyrir verkniðurröðun

Umfjöllunarefni þessarrar rigerðar eru nálgunaraðferðir fyrir verkniðurröðun. Aðferðirnar skoða næstu verk sem bíða niðurröðunar, mynda þaðan fjölda lausna með Monte Carlo útspilun (*e. rollouts*) og ákveða svo næsta verk til þess að raða. Rannsóknir á þessum aðferðum hafa hingað til einskorðast við röðunarreglur sem beina leit sinni að takmörkuðu svæði lausnarrúmsins. Aðferðirnar sem hér eru kynntar víkka leitarsvæði útspilanna með því að einblína á slembnar röðunarreglur. Fjórum nýjum reikniritum er lýst: styrktarreiknirit (*e. fortified rollout algorithm*), meðaltalsreiknirit (*e. average rollout algorithm*), tvinnreiknirit (*e. hybrid rollout algorithm*) og hlutfallsmarksreiknirit (*e. quantile rollout algorithm*) og liggur munur þeirra í því hvernig verkum er raðað eftir lok útspilanna. Aðferðirnar voru prófaðar á 600 mismunandi verkniðurröðunarverkefnum af þremur mismunandi stærðum. Allar nýju aðferðirnar fundu betri lausnir en önnur þekkt útspilsreiknirit. Hlutfallsmarksreikniritið fann bestu lausnirnar í heildina, sérstaklega fyrir stærri gerðir verkefna.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

How to allocate resources to jobs is one of the most important issues of production management. Most companies have limited amount of resources that they need to use for maximum productivity. In large companies with many different jobs, the task of allocating resources optimally to the different jobs quickly becomes impossible to solve manually. This is known as a scheduling problem.

The problem is how to allocate a number of jobs to a number of machines to meet a predefined criteria. Due to it's practical nature, scheduling problems have been widely researched since the industrial revolution of the 19th century. To begin with, simple rules, such as *first in first out*, were introduced to help production managers and many of these rules are still in use today. But these rules mostly give suboptimal solutions and in the middle of the 20th century mathematical formulations and algorithms for solving scheduling problems more accurately started to receive attention. The focus of this thesis will be on scheduling problems known as job-shop problems.

Job-shop problems are very difficult to solve to optimality and to account for this most methods used to tackle these problems are approximation methods, i.e. methods that are able to find good solutions in a reasonably short time period, without the requirement of optimality. These methods range from very simple rules that require little knowledge, to sophisticated ones that require detailed knowledge about the problem at hand.

Since optimal methods are too slow for practical use we will introduce new methods to approximate solutions to the job-shop problem. These new methods are based on machine learning methods used successfully for playing the game of Go and are called rollout algorithms. Rollout algorithms have previously been used by Bertsekas et al. (1997) and Duin and Voß (1999) for solving combinatorial optimization problems, by Runarsson et al. (2011) and Meloni et al. (2004) for the job-shop problem. In this thesis, rollout algorithms are expanded towards methods that resemble the reinforcement learning tactics introduced by Sutton and Barto (1998).

1.1 Motivation

The main difficulty of the job-shop problem is that as more machines and jobs are added to the problem its complexity increases exponentially and along with it, the computational time required to solve it. Despite increased computational capabilities of the last decades, larger job-shop problems are still extremely difficult to solve optimally and most of them are impossible to solve efficiently.

Due to the difficulty in solving to optimality, there exist many methods for finding good approximate solutions, ranging from very simple to complex and sophisticated methods. Approximation methods search for good solutions to the problem without the requirement of optimality. The most simple ones are not very accurate and the most sophisticated ones are usually knowledge based, i.e. they depend on special features of particular problems and do not necessarily give good results for other related problems.

Many approximation methods formulate the job-shop problem as a sequential decision problem, i.e. the task of successively choosing one of multiple available actions, each resulting in a different outcome. Consider a board game were we are in a particular state and wish to find the action that maximizes our chances of victory. This scenario is similar to the one we encounter with job-shop problems. We have a set of available jobs and wish to find which one to dispatch in order to maximize/minimize our objective function.

A popular method to find this action, when playing board games, is to play several random games (this is known as Monte Carlo rollouts), originating from the different available actions and pick the one that is the most likely to lead to victory. The aim of this thesis is to investigate whether Monte Carlo rollouts and rollout algorithms can be as successful in finding good solutions to job-shop problems, as they are in board games such as Go (Brügmann, 1993). Is it is possible to transfer the ideology of rollouts to deterministic optimization problems such as the job-shop problem? Can we get reasonable results for large, combinatorial, optimization problems using rollout algorithms?

1.2 Contribution

One of the greatest difficulties of designing algorithms for the job-shop problem, is that different problem instances vary significantly and so does the performance of the algorithms. The Monte Carlo rollouts used are random and do not depend on the particular problem at hand and should work for *any type of job-shop problem.*

This removes the necessity of the user to study the problem in detail before starting the solution process and, enabling non-experts to use the method. The new methods offers multiple strategies for implementation that finds good solutions for all kinds of different job-shop problems and it gives insight into how strategies work differently for problems of varying size.

The new methods and algorithms in the thesis are based on the new job-shop bandit (section 4.3). The *quantile rollout algorithm* (section 5.2.3) is most noteworthy but other methods and algorithms presented in the thesis are the *fortified rollout algorithm* (section 5.2.1), the *average rollout algorithm* (section 5.2.2), the *hybrid rollout algorithm* (section 5.2.4) as well as a new heuristic, *randomly chosen dispatch rule* (see section 5.1.2).

1.3 Overview

The job-shop problem is introduced in chapter two, its complexity and known methods for solving the problem are discussed. We follow up in chapter three with an introduction to rollout algorithms for combinatorial optimization problems, the mathematical framework they are used in and several rollout strategies. We also show how the job-shop problem can be framed in terms of rollout algorithms. In the fourth chapter the bandit problem is discussed together with several variants of it. The job-shop bandit, where the job-shop problem is framed as a bandit problem, is introduced. The fifth chapter describes strategies, methods and different versions of rollout algorithms for solving job-shop problems. In this chapter new methods for dealing with the job-shop problem are proposed and explained in detail. In the sixth chapter the experimental setup is explained and results are presented. The seventh chapter is a summary and discussion of the results. Conclusions and directions of future research are presented in chapter eight.

2 The Job-Shop Problem

The job-shop problem consists of a set of jobs that need to be processed on a set of machines. The problem assumes that a machine can only process one job at a time and that the processing of a job, referred to as an operation, cannot be interrupted. In an $n \times m$ job-shop problem, n jobs must be processed on m machines (Carlier and Pinson, 1989). If a job requires processing on a machine more than once the job-shop is said to be recirculating (Pinedo, 2008). The jobs are scheduled as a chain of operations where each operation needs to be processed during a given time period on each machine. The objective is to find the set of operations (a schedule) that has minimum length (Vaessens et al., 1994).

2.1 A Formal Statement of the Job-Shop problem

The following formal formulation of the job-shop problem is influenced by Vaessens et al. (1994) and Runarsson et al. (2011). A set $\mathcal{M}$ of m machines is given together with a set $\mathcal{J}$ of n jobs and a set $\mathcal{O}$ of $n * m$ operations. For each operation, $o \in \mathcal{O}$, there is a processing time, $p(o) \in \mathbb{N}$, a unique machine $M(o) \in \mathcal{M}$ where it is processed, and a unique job $J(o) \in \mathcal{J}$ to which it belongs. Hence each $o \in \mathcal{O}$ can be denoted by o_{ij} where i is the job and j the machine belonging to this operation.

The order of the jobs on the machines is specified in a permutation matrix $\sigma_{j,i} \in \mathcal{M}$, where $\sigma_{j,i}$ is the i-th machine that needs to process job j. The matrix σ decides the total ordering of operations for each job. It should be noted that there are no precedence constraints between operations of different jobs. Let $s(o_{j,i})$ be the starting time of job j on machine i. A schedule is a function, $S : \mathcal{O} \rightarrow \mathbb{Z}^+$, that for each operation o defines a start time $s(o)$. A feasible schedule must hold the following constraints:

$$s(o_{j,i}) \geq 0 \qquad \forall j \in \mathcal{J} \;\; i \in \mathcal{M} \tag{2.1}$$
$$s(o_{j,\sigma_{j,i}}) \geq s(o_{j,\sigma_{j,i-1}}) + p(o_{j,\sigma_{j,i-1}}) \qquad \forall j \in \mathcal{J} \;\; i \in \mathcal{M} \tag{2.2}$$
$$s(o_{j,i}) \geq s(o_{k,i}) + p(o_{k,i}) \text{ or } s(o_{k,i}) + p(o_{k,i}) \leq s(o_{j,i}) \tag{2.3}$$
$$\forall j,k \in \mathcal{J}, j \neq k, \;\; i \in \mathcal{M}$$

Constraint (2.1) states that the starting time of each job must be non-negative. Constraint (2.2) guarantees that all precedence constraints are satisfied. Constraint (2.3) makes sure that no machine can process two jobs at the same time. The length of a schedule, or the *makespan*, is the completion time of the last job and is defined as

$$C_{max} = \max_{o \in \mathcal{O}} s(o) + p(o) \tag{2.4}$$

The objective of the job-shop problem is to minimize the makespan

$$\min C_{max} = \max_{o \in \mathcal{O}} s(o) + p(o) \tag{2.5}$$

A feasible schedule is called *left-justified* if no job can start earlier without changing the processing order on any machine. If no job can start a operation without delaying another operation it is called *active* (Vaessens et al., 1994).

The most widely researched job-shop problems are deterministic, where the processing time is considered to be known and without uncertainties. Additional time required during processing, such as set-up times, is often implemented in the model as additional processing time. This is of course a simplification, since the processing time of a job in the real world is always stochastic. Despite this simplification, solving deterministic problems is a useful tool in scheduling to give insight in the workings of the job-shop (Pinedo, 2008).

2.2 Complexity of the Job-Shop Problem

The job-shop problem is extremely difficult to solve to optimality. The first examples of efficient methods that deal and solve scheduling problems are credited to Johnson

(1954) who developed an algorithm to solve a two machine flow shop that minimizes the maximum flow time. The idea of minimizing the makespan has since then been attributed to Johnson. His method was easily extended to different variations of simple scheduling problems (Jain and Meeran, 1998).

There exist algorithms that solve the job-shop problem optimally, but the worst case running time increases exponentially with respect to the size of the input except for a few special cases (e.g. Hefetz and Adiri (1982) show that job-shop problems with $m = 2$ and $p(o) = 1$ is such a case). Garey et al. (1976) showed that the job-shop problem is NP-hard for instances where $m \geq 3$ and $n \geq 3$. This means that it is highly unlikely that an algorithm with polynomial running time exists for these problems. A well known NP-hard problem is the traveling salesman problem. This problem can be looked upon as a special case of the job-shop problem with one machine ($m = 1$), i.e. the machine is the salesman and the jobs are the cities.

While many special cases of the job-shop problem have been solved to optimality there are for a given $n \times m$ instance $(n!)^m$ possible solutions to the problem. To put this into context, a 20×10 problem has $(20!)^{10} = 7.2651 \times 10^{183}$ possible solutions. That is more than the supposed age of the universe in microseconds (Jain and Meeran, 1998). The difficulty of the job-shop problem can also be seen in the fact that a 10×10 problem, proposed by Fisher and Thompson (1963), was not solved to optimality until 1986 (Vaessens et al., 1994).

Knowing that the job-shop problem is NP-hard and extremely difficult to solve is of no consolation for those faced with such a problem. But it might help that not all job-shop problems are equally hard to solve from a practical point of view. Some can be solved using dynamic programming or branch and bound methods (discussed in section 2.4.2). But these methods only produce satisfactory results if the problem is not too large.

Since it is not feasible to produce optimal solutions for larger problems in a reasonable time, considerable effort has been put into approximation methods, i.e. efficient methods that search for good solutions, without the requirement of optimality.

2.3 Approximation Methods for Solving the Job-Shop Problem

Heuristics, or dispatching rules as they are sometimes called, dispatch jobs according to predefined rules. There are several simple heuristics commonly used for solving the job-shop problem. Meta-heuristics modify complete solutions, usually found

from heuristics in search of better ones.

2.3.1 Heuristics

Heuristics are simple dispatch rules gained from experience and expertise. Most of these rules are dependent on the type of problem at hand and are in general not efficient for different types of problems. Each method dispatches a job, one at a time, based on the current state of all machines and jobs, to create a feasible and good schedule (Kawai and Fujimoto, 2005).

Popular dispatching rules include *shortest processing time* (SPT) where the job with the shortest processing time is dispatched next and *First Come First Serve* (FCFS) where the first job to enter the job-shop is dispatched next. Several well known dispatching rules are listed in table 2.1.

2.3.2 Meta-Heuristics

Meta-heuristics combine different heuristic methods and search among several schedules. The meta-heuristics search for better schedules mostly by manipulating a complete schedule repeatedly at each iteration. The family of meta-heuristics cover a large variation of techniques, from the very simple to sophisticated ones. The drawback of meta-heuristics is that they often depend on the properties of each problem for which they were developed. Implementing properties of a certain type of problem into the solution process helps the search for high-quality solutions, but when applied to a more general form of the problem it can prove to be less helpful or even counterproductive (Kawai and Fujimoto, 2005).

Popular meta-heuristics that have been used to solve the job-shop problem include simulated annealing (Laarhoven et al., 1992), tabu search (Pezzella and Merelli, 2000; Sun et al., 1995) and genetic algorithms (Croce et al., 1995).

Table 2.1: Popular heuristics and their description.

Heuristic	Description
Shortest Processing Time	The job that has the shortest processing time is dispatched next.
Processing time left	The job that has the least processing time left is dispatched next.
Machine Release Time	The job that generates the earliest release time on its machine is dispatched next.
LQUE	The job with the smallest difference between duedate and (processing time + tail) is dispatched next.
New Makespan	The job that adds least to the new makespan is dispatched next.
Release Time	The job that has the shortest release time is dispatched next.
Start Time	The job with the earliest possible start time is dispatched next.
Earliest Completion Time	The job with the earliest completion time is dispatched next.
Longest Processing Time	The job with the longest processing time is dispatched next.
First Come First Serve	First job arriving at a machine is dispatched next.
Random	A randomly chosen job is dispatched next.

2.4 Optimal Methods for Solving the Job-Shop Problem

There exist several methods for solving the job-shop problem to optimality. For smaller problems they work well but as the problem size increases, so does the computational time required. The most relevant exact methods are mixed integer programming and branch and bound.

2.4.1 Mixed Integer Programming

Mixed integer programming (MIP) is the mathematical formulation of a linear program with linear constraints and an objective function, with the additional constraint that some of the decision variables are integers. MIP formulations for the job-shop problem set up the constraints discussed in section 2.1. The main problem with MIP is that for most formulations, the number of integer variables scales exponentially and even when using more compact formulations they still require a large number of integer variables. This makes them difficult to solve to optimality (Jain and Meeran, 1998). For an overview of MIP formulations for the job-shop problem see Pan (1997).

Advances in computing technology and integer programming software of the past decade has made mathematical programming based scheduling receive more attention from researchers, even if they are not yet considered efficient for tackling large scheduling problems. MIP methods help researchers study special cases, which possess certain structures, that are solvable and there are often various partial relaxations of the constraints that can be useful (Pan and Chen, 2005). Since mathematical approaches have shown to be inadequate for solving job-shop problems arising in practice, more effort has been put into enumerative methods such as branch and bound (Pinedo, 2008).

2.4.2 Branch and Bound

Branch and bound is an enumerative method of solving combinatorial optimization problems. The main idea of branch and bound is to intelligently enumerate all feasible solutions of the problem, $\mathbf{S}$. To do this $\mathbf{S}$ is divided into subproblems, S_i, which is a subset of all feasible solutions of the optimization problem: $S_i \subseteq \mathbf{S}$. Branch and bound consists of three phases.

1. **Branching:** The process of branching is to divide all feasible solutions, $\mathbf{S}$, into subproblems, S_i for $i = 1, \ldots, n$ such that $\bigcup_{i=1}^{n} S_i = \mathbf{S}$. Branching is a recursive process so each S_i may be divided into further subproblems. The branching is represented as a branching tree where $\mathbf{S}$ is the root and S_i, for $i = 1, \ldots, n$ are its children (or branches).

2. **Bounding:** The process of bounding is to calculate a lower bound, LB, and an upper bound, UB, for all feasible solutions to a subproblem.

3. **Pruning:** If the lower bound, LB, of a subproblem is greater or equal to the

best upper bound, UB, this subproblem can not produce a better solution than the one already found and is discarded. If subproblems cannot be pruned, the branching must continue from current subproblems.

There are many ways to implement branch and bound algorithms and they mostly differ in the way the branching is done. When calculating lower bounds one must often choose between tight bounds, which may be computationally heavy, and wider bounds that are more computationally efficient. At each branching step, a choice must be made of which node to branch next. Common strategies are least-lower-bound-next, last-in-first-out and first-in-first-out. Which design strategy to use depends on the problem at hand, and its data (Brucker, 2007).

The most effective branch and bound methods for the job-shop problem are based on the so-called disjunctive graph model. For the job-shop problem, all operations of the same job are connected using conjunctive (directed) arcs and operations of different jobs are connected using disjunctive (undirected) arcs. When building a complete schedule precedence relations are fixed between operations by turning disjunctive arcs into conjunctive. A set of fixed disjunctions defines a feasible schedule if and only if every disjunctive arc has been fixed and the resulting graph is acyclic (Brucker, 2007).

At the beginning of the algorithm, the tree only contains one node, the root, for which no disjunctive arcs are fixed. The root represents all feasible solutions to this problem. Branching is started by fixing disjunctions. The corresponding disjunctive graph represents all feasible solutions, respecting these disjunctions. Every disjunctive graph can then be examined recursively in the same way. If a node only represents one solution (i.e. a complete solution) or it can be shown that the node does not contain an optimal solution, branching is stopped from that node.

2.5 Summary

This chapter introduced the job-shop problem. The problem was formally stated and its complexity discussed. Several approximation methods for finding solutions to the problem were introduced and heuristics and meta-heuristics were described briefly. Methods for finding the optimal solution of the job-shop problem were discussed, mixed integer programming and branch and bound.

In the next chapter, rollout algorithms for generating solutions to job-shop problems will be introduced.

3 Rollout Algorithms for Combinatorial Optimization

Rollout algorithms for combinatorial optimization developed by Bertsekas et al. (1997), or the equivalent pilot method developed by Duin and Voß (1999), are metaheuristic methods aimed at improving solutions of known heuristics. Rollout algorithms improve the performance of heuristics by sequential application of the heuristic. Rollout algorithms can be very useful when exact methods are too slow and solutions obtained by existing heuristics are not good enough.

3.1 A Framework for Rollout Algorithms

The framework used for rollout algorithms can be characterized by a finite set $\mathbf{O}$ of feasible solutions and a cost function $g(\mathbf{o})$. Each solution has K components of the form $\mathbf{o} = (o_1, o_2, \ldots, o_K)$, called the *path* of the solution. Our objective is to find the solution $\mathbf{o} \in \mathbf{O}$ that minimizes the function $g(\mathbf{o})$. It is possible to view the problem as a sequential decision problem, whereby the components of $(o_1 \ldots o_K)$ are selected one at a time. A solution containing the first k components, $(o_1, \ldots o_k)$ is a *partial solution* to the problem and is called a *k-solution*. A solution containing all K components is a *complete solution* and is called a K-solution (Bertsekas et al., 1997).

Let $J^*(o_1, \ldots o_k)$ be the optimal cost starting from a *k-solution*. Finding this optimal solution is seldom viable, due to the large computational effort required. This can be dealt with by replacing $J^*(o_1, \ldots o_k)$ with an approximation $\tilde{J}(o_1, \ldots o_k)$ and obtaining a suboptimal solution $(\tilde{o}_1, \ldots \tilde{o}_K)$ with the algorithm:

$$\tilde{o}_i = \arg\min_{o_i \in O} \tilde{J}(\tilde{o}_1, \ldots \tilde{o}_{i-1}, o_i), \quad i = 1, \ldots, K \tag{3.1}$$

The function $\tilde{J}$ is called a scoring function (Bertsekas et al., 1997).

Let us assume a heuristic algorithm $\mathcal{H}$, which when given a *k-solution* sequentially constructs a *K-solution*. This heuristic is called a *pilot heuristic* following Duin and Voß (1999). A typical step of a rollout algorithm is to apply a pilot heuristic, $\mathcal{H}$, from all components that are available for selection and move to the component that finds the best solution. The sequential version of $\mathcal{H}$ is called the *rollout algorithm based on* $\mathcal{H}$. A pseudocode for a typical rollout algorithm can be seen in algorithm 3.1 (Bertsekas et al., 1997).

Algorithm 3.1 A rollout algorithm for combinatorial optimization problems

Input: A k-solution $\mathbf{o}_k = (o_1, o_2, \ldots, o_k)$ and pilot heuristic $\mathcal{H}$
Output: A good complete solution.

repeat
  for all i where $o_i \in \mathcal{O}$ and $o_i \notin \mathbf{o}_k$ **do**
    $\tilde{J}_i = \mathcal{H}(\mathbf{o}_k, o_i)$
  end for
  $i^* \leftarrow \arg\min_i \tilde{J}_i$
  Operation o_{i^*} is added to the end of k-solution, $\mathbf{o}_k$
until complete solution found

3.2 Sequentially Improving Algorithms

A heuristic algorithm that generates the path $\mathbf{i} = (i, i_1, i_2, \ldots i_K)$ when starting with component i and the path $\mathbf{i}' = (i_1, i_2, \ldots, i_K)$ when starting with component i_1, is called *sequentially consistent*. If the pilot heuristic, $\mathcal{H}$, of a rollout algorithm is sequentially consistent then the rollout algorithm is terminating and has the following property:

$$H(i_1) \geq H(i_2) \geq \ldots \geq H(i_K) \tag{3.2}$$

where $H(i_1)$ is the cost of $\mathcal{H}$ starting from component i_1. If the algorithm has this property the next k-solution is guaranteed to be no worse than the current and the rollout algorithm is said to be *sequentially improving*.

If the pilot heuristic is sequentially inconsistent, it is possible to modify the rollout algorithm to make it sequentially improving. If no better solution is found from the available components, it follows the path from the current component and thus the algorithm stays sequentially improving. This is called the *extended rollout algorithm*.

If the rollout algorithm is not sequentially improving, it is possible to modify it so that from the first component it always follows the best path found so far. It always keeps the best path starting from the first component and does not leave the path until a improved path is found. If/When a better path is found it switches paths. This is called the *fortified rollout algorithm* and will be discussed further in chapter 5.

If the pilot heuristic is sequentially inconsistent, then it is possible that the rollout algorithm generates a worse solution than the solutions generated by the pilot heuristic. This can be corrected by a minor modification of the rollout algorithm. When running the algorithm, one can generate several solutions and upon termination choose the solution that gives minimal cost. This is called the *optimized rollout algorithm*. If the pilot heuristic is sequentially consistent all rollout algorithms discussed coincide (Bertsekas et al., 1997).

3.3 A Rollout Algorithm for the Job-Shop Problem

A rollout algorithm based on the job-shop problem, as presented by Runarsson et al. (2011) and Meloni et al. (2004), uses the property that the problem can be defined as a sequential decision problem where the solution is built in stages. The components, in this case the operations, are chosen one at a time to build a complete and feasible schedule.

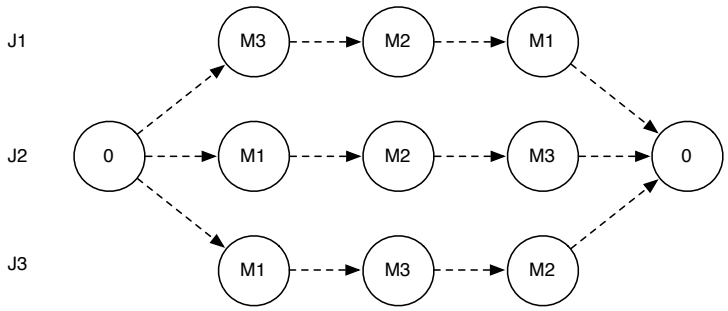

Figure 3.1: Graph for a 3×3 problem.

The problem instance is encoded as a graph. An example is given in figure 3.1 for a problem with 3 jobs and 3 machines (3×3). To the left and right there are dummy nodes labeled with 0 that do not belong to any job and this is where the algorithm starts and ends respectively. Every row of nodes (excluding the two dummy nodes) belongs to a job, the top row belongs to job 1, the next to job 2 and so on. Every node corresponds to a machine, and is marked with an M, so that the first node in the first row in figure 3.1 belongs to job 1 and machine 3, or operation o_{13} (recall that

o_{ji} is job j processed on machine i). The arcs represent the precedence constraints, a node cannot be visited until all the nodes that have a arc pointing at it have been processed. The second node in the first row is operation o_{12} and the arc denotes that it cannot be processed until after o_{13} is finished. The precedence matrix, σ, for this problem is:

$$\sigma = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$

The rollout algorithm for the job-shop problem uses a look-ahead procedure to choose the next job. For every k-solution the number of possible nodes to visit next are at most the same as the number of jobs. Recall that the pilot heuristic must be able to create a feasible K-solution from any given k-solution, so at each available node, the underlying pilot heuristic is used and a feasible K-solution is found and its makespan is calculated. The node that gave the best K-solution is then chosen to be processed next. From the new $(k+1)$-solution the procedure is repeated, all available nodes tested and the best one found. This is repeated until all nodes have been visited and a complete and feasible K-solution has been found. A pseudocode for a rollout algorithm for solving the job-shop problem is given in algorithm 3.2. The vector $\mathbf{t}$ keeps a record of how many machines the jobs have visited, t_j is the number of machines job j has visited so far.

Algorithm 3.2 A rollout algorithm for the job-shop problem

Input: Partial solution S_0, precedence matrix σ, J, number of jobs m, $\mathbf{t} = (t_1, \ldots, t_n)$ and the cost of heuristic $\mathcal{H}$ from partial solution S_0 starting from job j is $\mathcal{H}(S_0, j)$.
Output: A complete and feasible schedule.

repeat
  for all $j \in J$ **do**
    if $t_j \leq m$ **then**
      $r_j = \mathcal{H}(S_0, j)$
    end if
  end for
  $j = \arg\min_j r_j$
  Operation $o_{j,\sigma_{j,t_j}}$ is added to the end of solution S_0
  $t_j \leftarrow t_j + 1$
until all operations have been assigned

An illustration of the procedure, for the 3×3 example is shown in figure 3.2. The

current node is shown in bold, the available nodes in stripes and the visited nodes in grey. The algorithm starts at the dummy node on the left labeled with 0. From the dummy node, three operations can be selected: $\{o_{13}, o_{21}, o_{31}\}$. For each of these operations a rollout is performed using the underlying heuristic and the node resulting in the best schedule, is chosen to be visited next. Assuming that the operation chosen is o_{13}, we then move to the node associated with o_{13}, remove that operation from the set of available operations and add the next operation of job 1 that is now available for allocation, o_{12}. The set of available operations is now $\{o_{12}, o_{21}, o_{31}\}$. The next two iterations are shown in figure 3.2. The procedure is repeated until all nodes have been visited and all the operations processed. The final makespan is then calculated.

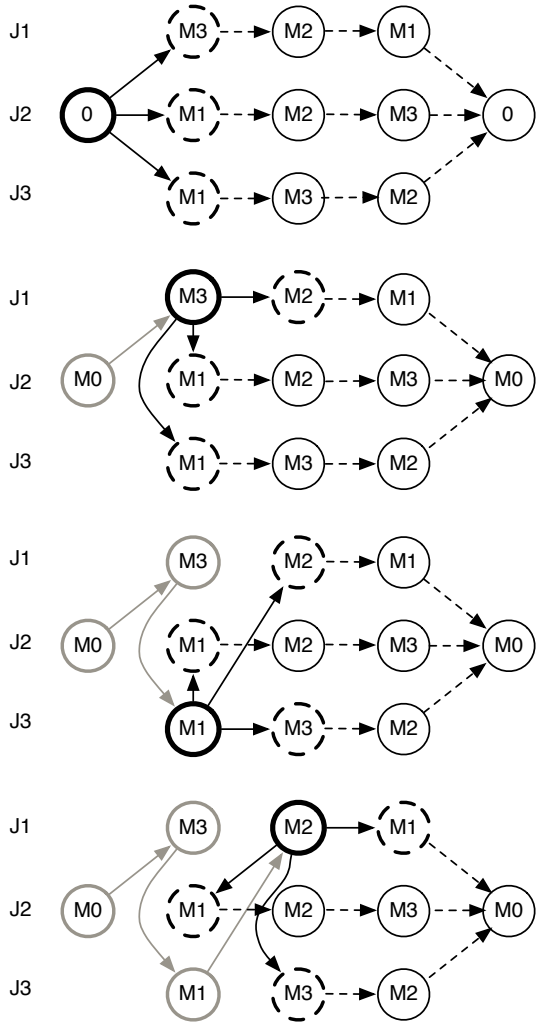

Figure 3.2: Graph of the 3×3 example. The first steps of the rollout algorithm using a look-ahead feature are shown. The current node is shown in bold, available nodes in stripes and visited nodes in grey.

3.4 Summary

In this chapter the fundamentals of rollout algorithms were explained and how they can be used to solve sequential decision problems. The chapter also described sequentially improving algorithms, together with some variations of the rollout algorithm. The last section described how the job-shop problem can be viewed as a sequential decision problem and how it can be solved with rollout algorithms.

In the next chapter the so-called bandit problem is explained as well as the job-shop bandit, were the rollout algorithm for the job-shop problem is expanded and set up as a bandit problem.

4 Bandit Algorithms

The K-armed bandit can be considered as a decision problem where one is faced with a hypothetical slot machine with K levers. Each lever gives a payoff coming from a stationary but unknown probability distribution (Alpaydin, 2010). The goal is to find out which lever to pull in order to maximize the expected cumulative payoff received over a series of n-trials. To solve the problem a balance must be found between exploration (pulling the lever to find out the reward) and exploitation (pulling the best known lever repeatedly) (Streeter and Smith, 2006).

4.1 The Conflict Between Exploration and Exploitation

Each lever has an expected reward or mean reward called the *true value* of that lever. If the value of each lever is known, solving the bandit problem is trivial and the lever with the highest value is always chosen. Since it is assumed that the distribution associated with each lever are not known with certainty, it is necessary to keep an estimation of the value of the different levers.

The lever that has the greatest estimated value is called the *greedy* lever. If the greedy lever is chosen it is called an *exploitation* action since the current knowledge of the lever values are exploited. If a non-greedy lever is chosen it is called an *exploration* action since the environment is being explored to improve the estimates of the lever values.

To get the maximum possible reward for a particular move, it is best to choose an exploitation lever. But in order to maximize the cumulative reward, for a fixed number of trials, it may often be better to choose an exploration lever since it gives a better estimate of the value of the non-greedy levers. Due to the uncertainty in the non-greedy lever values, one of them might actually have a better expected return than the greedy lever. So even if the reward is lower in the short run, a better estimate might be found of a non-greedy lever and by exploiting that lever the total reward might be higher in the long run. Whether it is better to exploit or explore depends on a number of factors such as the uncertainty of the lever values and the

number of remaining pulls. Due to the impossibility of exploring and exploiting at the same time one often refers to the *conflict* between exploration and exploitation (Sutton and Barto, 1998).

The conflict between exploration and exploitation is critical. A balance has to be found between exploring the environment for gathering statistics and searching for profitable actions, while taking the best action as often as possible and keeping the computational time at a minimum (Auer et al., 2002).

4.2 Evaluating Levers

When solving the K-armed bandit the means by which the levers are evaluated is critical. The methods vary from very simple to complex methods. The *true value* of a lever, $a = 1, 2, \ldots K$, is denoted $Q(a)$ and the estimated true value of lever a, after t pulls, is denoted $Q_t(a)$ (Sutton and Barto, 1998). The most straightforward method to evaluate the true value of a lever is to calculate the mean reward received so far from that lever. If after t plays, lever a has been chosen n_a times, giving the rewards $r_1, r_2, \ldots, r_{n_a}$ then

$$Q_t(a) = \frac{r_1 + r_2 + \ldots r_{n_a}}{n_a} \tag{4.1}$$

If $n_a = 0$, then $Q_t(a)$ is defined at some initial value, such as $Q_0(a) = 0$. From the law of large numbers, $Q_t(a) \to Q(a)$ when $t \to \infty$. This method is called the sample-average method.

The intuitive choice of a lever is to choose lever $a^* = \arg_a \max Q_t(a)$, but as discussed in section 4.1 this always exploits current knowledge of the levers and due to uncertainties in the estimates of the non-greedy lever values one of them might have a higher true value and alternative methods are therefore needed. To take this into consideration, so-called ϵ-greedy methods are commonly used, where a non-greedy lever is chosen during a trial with probability ϵ. This has been shown to be far more effective, for most problems, than choosing constantly the greedy lever (Sutton and Barto, 1998).

The choice of ϵ depends on the task at hand. If the variance of the reward is high, it is best to use a high value of ϵ, to be able to evaluate the levers true value accurately. If the variance is low, it is better to have a small degree of exploration (low value of ϵ) since our estimate of the true value becomes quite accurate. If the variance of the

rewards are zero, it is best to always choose the greedy lever ($\epsilon = 0$), since it will always give us the best outcome. Another commonly used strategy is to decrease ϵ with time. A high value of ϵ in the early stages leads to reliable estimates of the levers and with time the aim is to exploit those values more often so ϵ is lowered to allow more exploitation (Sutton and Barto, 1998).

4.3 The Job-Shop Bandit

To expand the framework introduced in section 3.3 the job-shop problem can be considered as a version of the K-armed bandit problem, where at each node the jobs available for processing next are the levers. The best job to dispatch at a given stage is not known so all available jobs are evaluated, using rollouts.

Instead of using a sequentially consistent heuristic during the rollout, the focus will be on sequentially inconsistent algorithms. This approach is new and has not been taken to the authors knowledge before. The most simple sequentially inconsistent heuristic is the random heuristic. The random heuristic dispatches jobs for processing at random. Since the random heuristic is sequentially inconsistent, every rollout will produce a different solution. By allocating a number of trials for every job, using known exploration and exploitation strategies, it is possible to generate several schedules with the random heuristic originating from the same job. By considering the distribution of these schedules for each job, it is possible to evaluate each job and choose the one showing the most promising results. The four stages of the job-shop bandit are:

1. Find all available jobs.

2. Allocate rollouts originating from these jobs.

3. Evaluate schedules found during rollouts.

4. Dispatch the most promising job and return to stage one.

When viewing the problem in this context, the exploration and exploitation strategies, i.e. how the rollouts are allocated to different jobs, is not the only matter of critical importance. Other important factors are which pilot heuristic to use when generating schedules during the rollout stage and to determine after the rollout stage, which job is deemed the most promising one. These issues will be discussed chapter 5.

4.4 The Max K-Armed Bandit

The max K-armed bandit is a version of the K-armed bandit were the goal is to maximize the maximum payoff received over a series of n trials. In any given trial it is possible to choose between K levers. This version of the bandit problem is useful when dealing with combinatorial optimization problems for which a number of search heuristics exists, like the job-shop problem (Streeter and Smith, 2006). The aim is to allocate trials to these heuristics to find out which is the most likely one to maximize the expected best single sample reward. The objective can be stated in the following way:

$$\max\left(\max_{i=1}^{K}\left(\max_{j=1}^{n_i} R_j(D_i)\right)\right) \tag{4.2}$$

where $R_j(D_i)$ is the reward of the j-th trial of lever i with reward distribution D_i (Cicirello and Smith, 2005). Cicirello and Smith (2005) assume that that the payoffs are drawn from the generalized extreme value distribution (GEV) and they derive an exploration strategy for allocating trials to different heuristics. The exploration strategy then follows the following distribution

$$P(\mathcal{H}_i) = \frac{\exp(R_i/T)}{\sum_{j=1}^{H}\exp(R_j/T)} \tag{4.3}$$

where R_i is an estimator of the expected maximum of a series of the trial heuristics, h_i, H is the number of trials allocated so far, and T is an exponentially decreasing temperature parameter. The aim is to increase the allocation of trials to the best levers double exponentially and to do this Cicirello and Smith (2005) propose to set $T = \exp(-j)$.

The assumption of a GEV distribution set by Cicirello and Smith (2005) may cause problems since it may be difficult to find a suitable distribution for the payoffs. Streeter and Smith (2006) propose a distribution free algorithm for solving the max K-armed bandit called Threshold ascent.

4.4.1 Threshold Ascent

It can be difficult to find a suitable distribution for the payoffs and Streeter and Smith (2006) assume that the payoff distribution does not belong to any specific

parametric family and make no formal assumptions at all about the underlying distribution. Their algorithm is called Threshold ascent. The algorithm works best when two criteria are satisfied:

1. There is a threshold, $t_{critical}$ such that for all $t > t_{critical}$ the lever most likely to yield a payoff greater than t is the lever most likely to yield a payoff greater than $t_{critical}$. This lever is called i^*.

2. As t increases beyond $t_{critical}$ the gap between the probability that lever i^* yields a payoff greater than t and the corresponding probability for other levers is growing.

The Threshold ascent algorithm for solving the max bandit problem sets a threshold $t_{critical}$ that varies over time. Initially it is set to zero and whenever s payoffs have been received that are greater than $t_{critical}$, the value of $t_{critical}$ is incremented. A pseudocode for Threshold ascent can be seen in algorithm 4.1.

Algorithm 4.1 Threshold ascent

Input: Number of levers K, parameter s, parameter δ, a small number Δ.
Output: Payoff of the different levers.

$t_{critical} \leftarrow 0$
for all i ={1,2,. . . , K} **do**
  $n_i \leftarrow 0$
end for
for all rollouts **do**
  while number of payoffs greater than threshold $\geq s$ **do**
    $t_{critical} \leftarrow t_{critical} + \Delta$
  end while
  $i^* \leftarrow \arg\max_i U(\frac{S_i}{n_i}, n_i)$
  Pull arm i^* and receive payoff R
  $n_i \leftarrow n_i + 1$
end for

Where $i = 1, 2, \ldots, K$ are the different levers, n is the total number of pulls, n_i is the number of times lever i has been pulled, S_i is the amount of rewards received by arm i above $t_{critical}$ and:

$$U(\mu, n) = \begin{cases} \mu + \frac{\alpha + \sqrt{2n\mu\alpha + \alpha^2}}{n} & \text{if } n > 0; \\ \infty & \text{otherwise.} \end{cases} \tag{4.4}$$

where $\alpha = \ln(\frac{2nk}{\delta})$ and δ is a parameter. The parameter s controls the tradeoff between exploration and exploitation. It is the amount of payoffs the strategy is

based on. If $s = 1$ then the policy is only dependent on how many pulls each lever has received and the lever that has had the least pulls so far is chosen. If $s \to \infty$ the policy equals the policy for the classical bandit problem (Streeter and Smith, 2006).

4.5 Summary

This chapter introduced the K-armed bandit problem. The conflict between exploration and exploitation and how levers can be evaluated was explained. A way of looking at rollout algorithms for the job-shop problem as bandit problems was introduced, referred to as the job-shop bandit. The max K-armed bandit problem was discussed together with threshold ascent algorithm for solving the max K-armed bandit.

In the next chapter a variety of rollout algorithms for solving the job-shop problem, based on the job-shop bandit, are introduced.

5 Rollout Algorithms for the Job-Shop Problem

The nature of the job-shop problem makes exact methods slow and easily implemented heuristics giving quality solutions are rare. By sequential application of a pilot heuristic it is possible to gather solutions of higher quality. Implementations of rollout algorithms, to solve the job-shop problem have been done by Meloni et al. (2004) and Runarsson et al. (2011) (chapter 3). Their results have shown that rollout algorithms for the job-shop problem improve the performance of the underlying heuristic. In this chapter the use of rollout algorithms to solve the job-shop problem is taken a step further.

This chapter is based on the job-shop bandit introduced in section 4.3. In the first section some useful pilot heuristics for the job-shop bandit will be discussed and rollout algorithms for the job-shop problem introduced. The last section describes several exploration policies. Some of these methods have been discussed in general terms in chapter 3 and 4 but are now described in detail for the job-shop problem. Others are new implementations first introduced here. These are the main contributions of the thesis.

5.1 Pilot Heuristics

The pilot heuristic is the method used for dispatching jobs during rollouts. A useful pilot heuristic for the job-shop bandit must generate a wide range of solutions since if it only generates a handful of solutions, it will converge to the original rollout algorithm of Bertsekas et al. (1997), discussed in chapter 3. In this thesis two pilot heuristics are used, the *random heuristic* and a new heuristic called *randomly chosen dispatch rule*.

5.1.1 Random Heuristic

The random heuristic chooses the next job at random from all the available jobs. This provides a wide range of solutions, some good and many bad. One advantage of the random heuristic is that it is completely knowledge free, i.e. no knowledge about the particular type of problem is necessary, making it possible to estimate the true value of choosing each possible job.

On the other hand the pure randomness makes the algorithm evaluate large areas of the solution space that are not likely to contain good solutions and many of the rollouts add little knowledge to the problem. But the random heuristic is easily programmed and the computational effort required is very low so even if more rollouts are required to find good solutions, compared to more sophisticated dispatching rules, the method may be able to produce reasonable results. The random heuristic is used for most experiments in this thesis.

5.1.2 Randomly Chosen Dispatch Rules

Another pilot heuristic which generates paths using multiple dispatching rules, $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_K$ is also studied. During the rollout, a dispatching rule is chosen at random at every node, i.e. every time a new operation is selected. Since the dispatching rules are selected randomly, the outcome is sequentially inconsistent and every rollout will generate a different schedule, giving us a way to estimate the true value of the different jobs.

The main advantage of randomly chosen dispatch rules, compared to the random heuristic, is that the solutions generated during rollouts are of a higher quality, so fewer rollouts should be required. On the other hand the computational effort required for rollouts is considerably higher than when using the random heuristic. The dispatching rules used are SPT, PL, MRT, LQUE, NM, RT, ST, ECT, LPT, FCFS (detailed descriptions of these dispatching rules may be found in table 2.1).

5.2 Policies for Dispatching Jobs

After the rollout stage a decision has to be made about which job to dispatch next. The aim is to follow the job showing the most promising results, but choosing which job is the most promising is not straightforward. In the following sections some policies for deciding which job to dispatch are discussed.

5.2.1 The Fortified Rollout Algorithm

The fortified rollout algorithm was introduced by Bertsekas et al. (1997) and is discussed in section 3.2. The fortified rollout algorithm is used when the pilot heuristic is not sequentially consistent (as with all the pilot heuristics in this thesis). The algorithm follows the best schedule that has been generated so far. If no better schedule is found from the next nodes the sequence found from the first node is followed until a better schedule has been found. This policy guarantees that the rollout algorithm stays sequentially improving, despite the sequential inconsistency of the pilot heuristic. A pseudocode for the fortified rollout algorithm can be seen in algorithm 5.1.

Algorithm 5.1 The fortified rollout algorithm for the job-shop problem

```
1:  Input: Processing times p, precedence matrix σ, J, number of machines m, number
    of jobs n and heuristic R.
2:  Output: A complete and feasible schedule.
3:
4:  S ← ∅
5:  for all j ∈ J do
6:     t_j ← 1
7:  end for
8:  V ← +∞
9:
10: for i = 1 → n * m do
11:    for all j ∈ J do
12:       if t_j < m then
13:          for k= 1, 2, ..., number of rollouts do
14:             [r_k, B] = R(S_{1→i}, j, p, σ)
15:             if r_k < V then
16:                S ← B
17:                V ← r_k
18:             end if
19:          end for
20:       end if
21:    end for
22:    j ← S_i
23:    t_j ← t_j + 1
24: end for
```

5.2.2 The Average Rollout Algorithm

When dispatching jobs after the rollout stage, it might seem intuitive to keep the algorithm sequentially improving. For this the fortified rollout algorithm is a good choice. But, for larger job-shop problems, the number of feasible solutions are extremely high (as discussed in section 2.2) and by using the fortified rollout algorithm there may be too much bias towards the best schedule found early. This may in return result in a low probability of visiting regions of greater solution quality.

To counter this, a new rollout algorithm is proposed, called the average rollout algorithm, that follows the job that finds the best schedule on average. This algorithm is not sequentially improving, so the final schedule found is not necessarily the best schedule found by the algorithm. To account for this, the final solution is the best solution found during rollouts. The advantage of the average rollout algorithm is that it should give a better estimate of the jobs scheduled first. This puts it in a favorable position in the solution space were good solutions can be found. A pseudocode for the average rollout algorithm is given in algorithm 5.2

Algorithm 5.2 The average rollout algorithm for the job-shop problem

Input: Processing times p, precedence matrix σ, J, number of machines m, number of jobs n and heuristic R.
Output: A complete and feasible schedule.

$S \leftarrow \emptyset$
for all $j \in J$ **do**
  $t_j \leftarrow 1$
end for
for $i = 1 \rightarrow n * m$ **do**
  for all $j \in J$ **do**
    $Q_j \leftarrow +\infty$
    if $t_j \leq m$ **then**
      $Q_j \leftarrow 0$
      for k$= 1, 2, \ldots,$ number of rollouts **do**
        $r_k = R(S, j, p, \sigma)$
        $Q_j \leftarrow Q_j + \frac{r_k - Q_j}{k}$
      end for
    end if
  end for
  $j \leftarrow \arg\min_j Q_j$
  $S_i \leftarrow j$
  $t_j \leftarrow t_j + 1$
end for

5.2.3 The Quantile Rollout Algorithm

The average rollout algorithm dispatches the job giving the most promising results on average, for all the rollouts. This means that a job that finds many solutions of low quality is discarded even if other parts of the solution space might contain high quality solutions. To counter this problem a new rollout algorithm is proposed called the quantile rollout algorithm.

Instead of gathering statistics of all rollouts and evaluating their average the quantile rollout algorithm discards all rollouts except those in the lower quantile. The quantile is found by dividing ordered data into q equally sized data subsets and is the data point which marks the boundary of these subsets. After completion of the rollout stage, the quantile rollout algorithm select the job that has schedules with the lowest average makespan in the lower quantile.

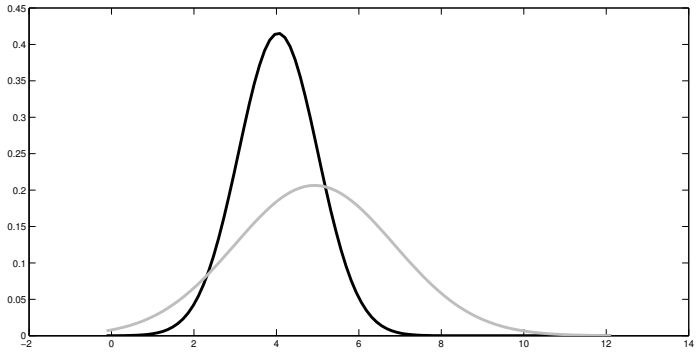

Figure 5.1: Two normal distributions where the one with the lower mean (black) is less likely to contain good schedules compared to the higher mean (grey).

The advantage of the quantile rollout algorithm is that it takes into account that the solutions for each job do not necessarily follow the same distribution. Consider figure **??**, the black distribution has a lower mean than the grey, but the grey distribution is more likely to have a high density of good solutions and therefore is more likely to yield good results. The quantile average rollout algorithm takes this into account by following the job that has the highest average of good solutions, not the highest average of good and bad solutions. In this study the lower *quartile* ($q = 4$) and *octile* ($q = 8$) are used.

5.2.4 A Hybrid Rollout Algorithm

A fourth rollout algorithm is proposed called the hybrid rollout algorithm. The hybrid rollout algorithm is a mixture of the fortified rollout and the average rollout

algorithms. The hybrid algorithm uses the average rollout algorithm for scheduling the first $(100 - \alpha)\%$ operations and the fortified rollout algorithm for the last $\alpha\%$ operations.

The hybrid rollout algorithm is intended to exploit the advantages of the two rollout algorithms for larger problems. When dispatching the first jobs, the number of feasible schedules is very high, but as more jobs are dispatched the solution space narrows down towards areas that are more likely to contain good solutions and the fortified rollout algorithm might become the better option. The main difficulty of the hybrid rollout algorithm is how to select the value of the α parameter. The value is likely to be dependent on the size of the job-shop problem.

5.3 Exploration Policies

As discussed in chapter 4, the choice of exploration policy during the rollout stage is of high importance. When using rollout algorithms to solve the job-shop problem there are various ways to explore the solution space. The choice of exploration does not only depend on the particular problem or it's size, but also on which rollout algorithm is in use.

5.3.1 Evenly Distributed Rollouts

A simple policy is to evenly distribute the rollouts on every available job. Since the objective of the rollouts is to gather as much information about every job, and not to maximize the cumulative reward, like in the classical job-shop problem, this exploration policy gives us the closest possible estimate of the true value of every available job. This is a good option when using the average rollout algorithm, since it gives the best estimate of the true value of the jobs. The total amount of rollouts per available job is then

$$N_j = \left\lfloor \frac{N}{n} \right\rfloor \tag{5.1}$$

where n is the number of jobs available for processing, N is the total amount of rollouts and N_j is the amount of rollouts for job j.

5.3.2 Threshold Ascent

The threshold ascent algorithm described in section 4.4.1 allocates more rollouts to the jobs that are finding the best schedules. A pseudocode for threshold ascent is given in algorithm 4.1. Since the reward of the job-shop problem is the makespan, records must be kept for the s best rollouts and from which job they originated. Our choice of which job to explore follows a distribution that depends on were the s best rollouts originated from. A job that consistently finds good solutions is more likely to be explored than jobs that did not find good solutions.

This exploration strategy is best when used with the fortified rollout algorithm since it explores jobs that are giving us good schedules more often. If used with the average rollout algorithm the uncertainties of the different jobs differ significantly and are not reliable for decision making.

5.4 Summary

In this chapter a variation of rollout algorithms that are based on the job-shop bandit were introduced. These were the fortified rollout algorithm, the average rollout algorithm, the quantile rollout algorithm and the hybrid rollout algorithm. We also discussed pilot heuristics and exploration policies. In the following chapter different variations of these methods are tested on job-shop problems.

6 Experimental Study

The performance of the proposed rollout algorithms, in solving the job-shop problem, is evaluated in this chapter. Since the new rollout algorithms are randomized search methods, they need to be tested multiple times on multiple problems of different dimensions in order to measure their efficiency and effectiveness. In the first section the set up of experiments and how the test problems are generated is discussed. In the second section the experimental results will be presented.

6.1 Experimental Setup

All the methods discussed in the thesis were tested on 600 variations of the job-shop problem. These problems are divided equally into three different $n \times m$ sizes: 6×6, 10×10 and 14×14. The problems were created using the methodology proposed by Taillard (1993) and solved to optimality using a branch and bound algorithm developed by Brucker (2007). The processing times of all the jobs on all the machines in all the problems are uniformly distributed between 1 and 200 and the sequencing of every job is a random permutation. Every job has to visit every machine once. The rollout algorithms were additionally tested on four well known 20×20 benchmark problems proposed by Yamada and Nakano (1992), for which no optimal solutions are known to the best of the authors knowledge.

Every problem was solved with 6 different number of rollouts: 100, 500, 1.000, 3.000, 5.000 and 10.000. Since the methods are randomized, every run gives a potentially different solution. To account for this 30 trials were carried out for every problem and every amount of rollout. Each problem was therefore solved 180 times in all.

The methods were programmed in `C` as a `mex`-file that are executable from `MATLAB`. The computations were performed on the University of Iceland computer cluster Sól.

6.1.1 Performance Criteria

When comparing the results of the different algorithms it is necessary to have an unbiased performance measure. Here the mean percentage error and (MPE) and average mean percentage error (AMPE) are used

$$\mathrm{MPE}_j = \frac{1}{n} \sum_{i=1}^{n} \frac{R_i - O_j}{O_j} \tag{6.1}$$

$$\mathrm{AMPE} = \frac{1}{t} \sum_{j=1}^{t} \mathrm{MPE}_j \tag{6.2}$$

where j is the problem index, n is the number of trials, R_i is the solution found with rollout algorithm in the i-th trial and O_j is the optimal solution of problem instance j. For our experiments $t = 200$ and $n = 30$. The results are tested for statistical significance with the Wilcoxon rank sum test.

6.2 Results

In this section the results of the experiments are presented. In the first subsections the results for the separate rollout algorithms are presented and in later sections the results of the different rollout methods are compared for each problem size and number of rollouts. The following statistics are displayed in tables:

1. **Min:** The *minimum* MPE found for a problem.
2. **Mean:** *AMPE* of all solutions found.
3. **Max:** The *maximum* MPE found for a problem.
4. **Stdev:** *Standard deviation* of the solutions found.
5. **Opt:** The percentage of solutions found that were *optimal*.

The results presented are based on 10.000 rollouts for each job dispatch decision made, unless otherwise indicated and shown in percentages. Abbreviations used to

describe the specific rollout algorithm and pilot heuristic when presenting results can be seen in table 6.1.

Table 6.1: Abbreviations.

Abbr.	Pilot	Exploration policy	Rollout algorithm	Section
DR_{ave}	Random dispatch rule	Evenly distributed	Average	6.2.2
DR_{for}	Random dispatch rule	Evenly distributed	Fortified	6.2.1
RH_{ave}	Random Heuristic.	Evenly distributed	Average	6.2.2
RH_{for}	Random Heuristic.	Evenly distributed	Fortified	6.2.1
HR	Random Heuristic.	Evenly distributed	Hybrid	6.2.3
RH_{for}^{TA}	Random heuristic	Threshold Ascent	Fortified	6.2.1
$RH_{1/4}^{Q}$	Random heuristic	Evenly distributed	Quantile	6.2.4

6.2.1 Results for the Fortified Rollout Algorithm

Rollouts were made using the fortified rollout algorithm which follows the best sequence found so far (see section 5.2.1). For the fortified rollout algorithm the pilot heuristics used were the random heuristic (see section 5.1.1) and using randomly chosen dispatch rule (see section 5.1.2). The exploration policies used were evenly distributed (see section 5.3.1) and threshold ascent (see section 4.4.1).

The results for the fortified rollout algorithm can be seen in table 6.2. For the 6×6 problems the mean error is close to zero and the optimal solution is found in most cases by all the methods. Using randomly chosen dispatch rules as the pilot heuristic finds the optimal solution slightly more often than other methods, but the difference is not statistically significant. For the 10×10 problems using randomly chosen dispatch rules gives the best results, with the other methods showing slightly worse results. For the 14×14 problems the same pattern was observed, using randomly chosen dispatch rules works best, but other methods are not lagging far behind. The standard deviation of the different methods is quite small, around $1 - 2\%$ of the optimal value.

Table 6.2: Results for the fortified rollout algorithm.

Problem	Method	min	mean	max	stdev	opt
6×6	DR_{for}	0.00	0.34	10.88	0.32	83.33
	RH_{for}	0.00	0.30	8.42	0.26	81.83
	RH_{for}^{TA}	0.00	0.30	8.35	0.29	81.87
10×10	DR_{for}	0.58	4.34	9.63	1.93	1.60
	RH_{for}	0.84	4.92	9.58	1.76	0.57
	RH_{for}^{TA}	0.59	4.84	9.17	1.85	0.58
14×14	DR_{for}	4.45	10.44	13.97	1.99	0.37
	RH_{for}	6.41	11.26	14.32	2.16	0.00
	RH_{for}^{TA}	5.95	11.19	14.03	1.76	0.22

6.2.2 Results for the Average Rollout Algorithm

Rollouts were made using the average rollout algorithm where the job that has the best average of solutions (see section 5.2.2) is followed. For the average rollout algorithm, the methods tested were the same as for the fortified rollout algorithm. The pilot heuristics used were the random heuristic and random dispatching rules. The exploration policies used were evenly distributed and threshold ascent.

The results for the average rollout algorithm can be seen in table 6.3. For the 6×6 and 10×10 problems the performance is inferior to the fortified rollout algorithm (table 6.2). Using randomly chosen dispatch rules again gives slightly better results for these problem sizes. For the 14×14 problems, the average rollout algorithm performs better than the fortified rollout algorithm on average. The difference between random heuristic and randomly chosen dispatch rules is statistically insignificant for this problem size. The optimal solution is not found except in a very few cases.

Table 6.3: Results for the average rollout algorithm.

Problem	Method	min	mean	max	stdev	opt
6×6	DR_{ave}	0.00	3.35	18.35	0.96	22.63
	RH_{ave}	0.00	4.62	16.19	1.30	14.77
	RH_{ave}^{TA}	1.32	6.71	17.46	3.84	8.10
10×10	DR_{ave}	1.03	6.45	12.93	1.87	0.28
	RH_{ave}	3.12	7.42	12.04	2.21	0.08
	RH_{ave}^{TA}	4.99	10.42	15.38	3.19	0.05
14×14	DR_{ave}	3.70	9.45	14.38	1.91	0.28
	RH_{ave}	4.11	9.17	12.71	1.95	0.00
	RH_{ave}^{TA}	6.20	13.29	17.22	2.67	0.18

6.2.3 Results for the Hybrid Rollout Algorithm

Since the average rollout algorithm is producing better results than the fortified rollout algorithm for large problems with a large solution space, a new method is proposed called the hybrid rollout algorithm. With the hybrid rollout algorithm the first $(100-\alpha)\%$ of operations are chosen with the average rollout algorithm but the other $\alpha\%$ of operations are chosen with the fortified rollout algorithm (see section 5.2.4 for details).

The hybrid rollout was only tested with the random hueristic due to the computational time required to use randomly chosen dispatch rule. The results for the hybrid rollout algorithm used with the random heuristics can be seen in table 6.4. For smaller problems (6×6 and 10×10) the hybrid rollout algorithm performs best when $\alpha = 30$. For 14×14 problems $\alpha = 40$ and $\alpha = 30$ give similar results. The hybrid rollout algorithm only managed to find the optimal solution in very few runs.

The hybrid rollout algorithm manages to find better schedules (on average) than the average and fortified rollout algorithm (see table 6.2 and 6.3) for larger problem instances. For smaller instances it performs worse than the fortified rollout algorithm, but similarly to the average rollout algorithm.

Table 6.4: Results for the hybrid rollout algorithm.

Problem	α	min	mean	max	stdev	opt
6×6	10	0.00	4.66	15.86	1.29	15.12
	20	0.00	4.70	15.94	1.32	14.63
	30	0.00	4.37	16.69	1.27	16.22
	40	0.00	4.67	16.22	1.36	14.95
10×10	10	2.30	7.38	11.81	2.20	0.08
	20	2.39	7.37	11.77	2.21	0.05
	30	1.04	5.91	11.27	2.09	0.33
	40	2.77	6.71	11.13	2.20	0.15
14×14	10	3.92	9.19	12.16	1.59	0.43
	20	4.31	9.13	12.36	1.37	0.35
	30	3.49	7.92	11.04	1.74	0.00
	40	3.61	7.88	11.17	1.62	0.30

6.2.4 Results for the Quantile Rollout Algorithm

The quantile rollout algorithm follows the job that has the best average of solutions in the lower quantile (see section 5.2.3). Two separate quantiles were tested, quartile

($q = 4$) and octile ($q = 8$), i.e. the ordered solutions were divided into 4 and 8 equally large subsets and the job with the lowest makespan average in the lowest quartile or octile was chosen to be processed next. Results for the quantile rollout algorithm are listed in table 6.5.

Table 6.5: Results for the quantile rollout algorithm.

Problem	Quantile	min	mean	max	stdev	opt
6×6	Octile	0.00	0.37	8.41	0.24	77.32
	Quartile	0.00	0.37	8.46	0.24	76.95
10×10	Octile	0.22	3.83	7.89	1.19	1.20
	Quartile	0.20	3.86	7.78	1.16	1.50
14×14	Octile	3.17	6.99	9.85	1.40	0.33
	Quartile	3.40	6.81	10.14	0.98	0.25

For the 6×6 and 10×10 problems there is no statistical difference between choosing the lower quartile or octile. For the 14×14 problems using the quartile gives slightly better results on average but the difference is still not statistically significant. The optimal solution, of the 14×14 problems, is only found in a few cases but the method produces results on average that are not very far off from the optimal solution.

6.2.5 Results for Threshold Ascent

Threshold ascent depends on two parameters, s and δ. The parameter s is the number of solutions that are kept and the next exploration move is chosen from. Experiments were made for the following values $s = 30, 50, 70, 100, 200$ and $\delta = 0.1$ as is done by (Streeter and Smith, 2006). Experiments with threshold ascent were only performed using the random heuristic. The reason for this is that it requires less computational effort. Experiments with Threshold ascent were performed using the fortified rollout algorithm and average rollout algorithm. Experiments using the average rollout algorithm did not produce satisfactory results and are not presented. The reason for the poor results of the average rollout algorithm is thought to be that the bias of allocating too many rollouts to the jobs that are finding the best solutions leads to bad estimates of the true value of the other jobs. This might lead to dispatching based upon unreliable job values. The results for threshold ascent as an exploration policy with the fortified rollout algorithm can be seen in table 6.6.

For the 6×6 problem instances, threshold ascent performs extremely well, finding the optimal solution in $81 - 82\%$ of the runs. As the problem size gets larger it is still able to find the optimal solution but only in very few instances. The mean is very low for the 6×6 and 10×10 problems but quite high for the 14×14 problems.

The value of s did not make a statistical difference for the results.

Table 6.6: Results when using threshold ascent as exploration policy.

Problem	s	min	mean	max	stdev	opt
6×6	30	0.00	0.30	8.35	0.29	81.87
	50	0.00	0.28	8.40	0.26	82.13
	70	0.00	0.27	8.35	0.23	82.43
	100	0.00	0.27	8.33	0.24	82.57
	200	0.00	0.28	8.33	0.26	82.20
10×10	30	0.59	4.84	9.17	1.85	0.58
	50	0.51	4.85	9.30	1.75	0.53
	70	0.66	4.78	8.94	1.74	0.47
	100	0.74	4.80	9.45	1.75	0.55
	200	0.59	4.73	8.99	1.70	0.58
14×14	30	5.95	11.19	14.03	1.76	0.22
	50	5.71	11.18	13.93	2.03	0.13
	70	6.13	11.10	14.45	2.45	0.15
	100	5.83	11.18	14.32	1.59	0.15
	200	5.50	11.07	14.12	2.05	0.22

6.2.6 Randomly Chosen Dispatch Rule

Experiments were made where the heuristic used to select the next job during rollouts was chosen at random from the list in table 2.1. The results are shown in table 6.7. For the 6×6 and 10×10 the fortified rollout algorithm produced better results, but for the 14×14 problems the average rollout algorithm is outperforming the fortified rollout algorithm. It is worth noting that this methodology requires considerably more computational time than when the random heuristic is used as the pilot heuristic.

Table 6.7: Results when using randomly chosen dispatch rules as exploration policy.

Problem	Method	min	mean	max	stdev	opt
6×6	DR_{ave}	0.00	0.34	10.88	0.32	83.33
	DR_{for}	0.00	3.35	18.35	0.96	22.63
10×10	DR_{ave}	0.58	4.34	9.63	1.93	1.60
	DR_{for}	1.03	6.45	12.93	1.87	0.28
14×14	DR_{ave}	4.45	10.44	13.97	1.99	0.37
	DR_{for}	3.70	9.45	14.38	1.91	0.28

6.2.7 Results for Different Number of Rollouts

Rollouts were made using six different budgets of rollouts for all the methods: 100, 500, 1.000, 3.000, 5.000 and 10.000. Table 6.8, shows the average mean percentage error (AMPE) for all the methods and all the different problem sizes. As expected the AMPE decreases with increasing amount of rollouts. The method giving the best results for a smaller amount of rollouts is when randomly chosen dispatch rules are used as the pilot heuristic, but the difference is not statistically significant. This method is very computationally demanding compared to the random heuristic.

For the 6×6 problems the results do not improve significantly when increasing the rollout number above 3.000 rollouts. For the 10×10 problems there is an improvement going above 3.000 rollouts but the gain is limited. For the 14×14 problems, the gain is significant and better results might be obtained by going above 10.000 rollouts.

Table 6.8: Average mean percentage error for different number of rollouts.

		Number of rollouts					
Problem	Method	100	500	1,000	3,000	5,000	10,000
6×6	DR_{ave}	4.27	3.52	3.44	3.36	3.34	3.35
	DR_{for}	2.47	1.13	0.80	0.52	0.44	0.34
	RH_{ave}	6.45	5.14	4.90	4.71	4.67	4.62
	RH_{for}	2.27	1.16	0.87	0.49	0.38	0.30
	HR	6.05	4.75	4.60	4.43	4.46	4.37
	RH^{TA}_{for}	2.24	1.13	0.86	0.49	0.40	0.30
	$RH^{Q}_{1/4}$	2.41	1.27	0.96	0.62	0.50	0.36
10×10	DR_{ave}	10.26	7.88	7.31	6.76	6.62	6.45
	DR_{for}	13.50	9.75	8.73	7.78	7.55	7.42
	RH_{ave}	10.39	7.50	6.57	5.38	4.96	4.34
	RH_{for}	9.13	7.33	6.64	5.77	5.41	4.92
	HR	12.68	8.43	7.36	6.47	6.11	5.91
	RH^{TA}_{for}	9.08	7.23	6.61	5.72	5.33	4.84
	$RH^{Q}_{1/4}$	10.73	6.55	5.64	4.50	4.16	3.83
14×14	DR_{ave}	16.07	12.13	11.02	10.00	9.71	9.45
	DR_{for}	18.06	14.68	13.46	11.82	11.22	10.44
	RH_{ave}	19.69	14.17	12.25	10.32	9.70	9.17
	RH_{for}	15.77	14.04	13.33	12.28	11.76	11.26
	HR	19.56	13.89	11.74	9.49	8.68	7.92
	RH^{TA}_{for}	15.92	14.01	13.27	12.24	11.84	11.19
	$RH^{Q}_{1/4}$	18.43	11.91	10.17	8.07	7.47	6.93

6.2.8 Results for Different Problem Sizes

In tables 6.9, 6.10 and 6.11 the results of all tested methods for problem sizes 6×6, 10×10 and 14×14 are presented. For threshold ascent, the hybrid rollout algorithm and quantile rollout algorithm only the best results from the different experiments are displayed.

For the 6×6 problems the fortified rollout algorithm is giving the best results (table 6.9). Threshold ascent with the random heuristics has the lowest error, a 0.28% on average. The optimal value is found most often using the fortified rollout algorithm with randomly chosen dispatch rules. Using the fortified rollout algorithm with an evenly distributed exploration strategy or using randomly chosen dispatching rules as the pilot heuristic are not far behind threshold ascent. The hybrid rollout algorithm is giving better results than the average rollout algorithm for this problem size.

Table 6.9: Results for the 6×6 problems.

Problem	Method	min	mean	max	stdev	opt
6×6	DR_{ave}	0.00	3.35	18.35	0.96	22.63
	DR_{for}	0.00	0.34	10.88	0.32	83.33
	RH_{ave}	0.00	4.62	16.19	1.30	14.77
	RH_{for}	0.00	0.30	8.42	0.26	81.83
	HR	0.00	4.37	16.69	1.27	16.22
	RH^{TA}_{ave}	1.32	6.63	17.46	3.84	8.10
	RH^{TA}_{for}	0.00	0.28	8.35	0.29	81.87
	$\mathsf{RH}^{Q}_{1/4}$	0.00	0.37	8.46	0.24	76.95

Results for the 10×10 problems can be seen in table 6.10. For this problem size, the quantile rollout algorithm is giving the best results with a 3.86% error on average from the optimal solution. The fortified rollout algorithm using randomly chosen dispatch rules is the second best and the fortified rollout algorithm using the random heuristic is the third best. It is worth to note that the hybrid rollout algorithm performs better than the fortified rollout algorithm with an evenly distributed random heuristic.

Table 6.10: Results for the 10×10 *problems.*

Problem	Method	min	mean	max	stdev	opt
10×10	DR_{ave}	0.58	4.34	9.63	1.93	1.60
	DR_{for}	1.03	6.45	12.93	1.87	0.28
	RH_{ave}	3.12	7.42	12.04	2.21	0.08
	RH_{for}	0.84	4.92	9.58	1.76	0.57
	HR	1.04	5.91	11.27	2.09	0.33
	RH_{ave}^{TA}	4.99	10.42	15.38	3.19	0.05
	RH_{for}^{TA}	0.59	4.84	9.17	1.85	0.58
	$RH_{1/4}^{Q}$	0.20	3.86	7.78	1.16	1.50

For the 14×14 problems (see table 6.11) the average rollout algorithm performs better than the fortified rollout algorithm. The method showing the best results here is the quantile rollout algorithm with a 6.81% error on average from the optimal solution and the hybrid rollout algorithm with an AMPE of 7.92%. For this problem size using randomly chosen dispatch rules is extremely computationally demanding compared to the other methods. For this larger problem it is worth noting that the threshold ascent exploration strategy, which showed promising results for 6×6 and 10×10 problems is not performing well, giving the highest percentage error.

Table 6.11: Results for the 14×14 *problems.*

Problem	Method	min	mean	max	stdev	opt
14×14	DR_{ave}	4.45	10.44	13.97	1.99	0.37
	DR_{for}	3.70	9.45	14.38	1.91	0.28
	RH_{ave}	4.11	9.17	12.71	1.95	0.00
	RH_{for}	6.41	11.26	14.32	2.16	0.00
	HR	3.49	7.92	11.04	1.74	0.00
	RH_{ave}^{TA}	6.20	13.29	17.22	2.67	0.18
	RH_{for}^{TA}	5.95	11.19	14.03	1.76	0.22
	$RH_{1/4}^{Q}$	3.40	6.81	10.14	0.98	0.25

6.2.9 Results for the Yamada 20×20 Problems

Yamada and Nakano (1992) proposed four different 20×20 benchmark job-shop problems. No optimal solution has been proven for these problems and the results are compared to best known solutions given by Banharnsakun et al. (2011). The results are presented in table 6.12. It can be seen that the quantile average rollout algorithm produces the best results for all four problems, the hybrid rollout algorithm produces the second best results. The same pattern was seen for the 14×14

problems in table 6.11. For these problems the average rollout algorithm also outperforms the fortified rollout algorithm. When using the quantile rollout algorithm, dividing the solutions into four different datasets and estimating the lower quartile, results in better schedules than dividing into eight datasets and estimating the lower octile. The difference between the two settings has increased compared to what was observed for the 14×14 problems in table 6.5.

Table 6.12: Results for the 20×20 problems.

Problem	Method	min	mean	max	stdev	% best
yn01	DR_{ave}	11.37	17.11	26.35	3.07	0.00
	DR_{for}	6.98	9.97	12.27	1.21	0.00
	RH_{ave}	7.55	10.68	12.73	1.39	0.00
	RH_{for}	12.16	16.88	19.82	1.83	0.00
	HR	7.77	9.70	12.05	1.13	0.00
	RH^{TA}_{for}	11.60	17.11	22.52	2.41	0.00
	$RH^{Q}_{1/4}$	6.76	8.48	10.59	0.95	0.00
	$RH^{Q}_{1/8}$	7.66	9.77	12.61	1.35	0.00
yn02	DR_{ave}	13.09	18.27	26.84	2.68	0.00
	DR_{for}	7.92	10.20	11.77	1.14	0.00
	RH_{ave}	8.14	11.20	14.19	1.45	0.00
	RH_{for}	12.65	17.95	22.00	1.90	0.00
	HR	5.72	10.18	13.97	1.64	0.00
	RH^{TA}_{for}	13.64	19.03	22.22	1.84	0.00
	$RH^{Q}_{1/4}$	6.82	9.72	13.64	1.45	0.00
	$RH^{Q}_{1/8}$	7.92	10.72	13.53	1.43	0.00
yn03	DR_{ave}	11.98	17.88	22.62	2.53	0.00
	DR_{for}	7.73	10.65	13.66	1.50	0.00
	RH_{ave}	7.95	10.96	13.77	1.51	0.00
	RH_{for}	12.99	17.81	22.40	1.98	0.00
	HR	8.17	10.45	14.11	1.48	0.00
	RH^{TA}_{for}	13.21	18.07	23.29	2.03	0.00
	$RH^{Q}_{1/4}$	6.05	9.00	11.09	1.21	0.00
	$RH^{Q}_{1/8}$	7.05	9.79	13.21	1.47	0.00
yn04	DR_{ave}	13.43	18.37	23.45	2.26	0.00
	DR_{for}	7.33	11.39	14.26	1.72	0.00
	RH_{ave}	9.50	11.48	14.67	1.13	0.00
	RH_{for}	12.60	17.59	22.31	2.03	0.00
	HR	8.06	11.95	14.98	1.63	0.00
	RH^{TA}_{for}	13.53	17.01	21.90	1.92	0.00
	$RH^{Q}_{1/4}$	6.71	9.77	13.02	1.52	0.00
	$RH^{Q}_{1/8}$	6.10	10.62	12.81	1.44	0.00

6.3 Computational Time Comparisons

Using rollout algorithms for the job-shop problem is computationally demanding. What matters most, in terms of computational effort, is the problem dimension and computational requirements of the pilot heuristic. In the experiments two pilot heuristics were used; the random heuristic and randomly chosen dispatch rule. In table 6.13 the average computational time it took to solve one problem, using 10.000 rollouts, is displayed for the three problem sizes. Using randomly chosen dispatch rule requires more computational time than the random heuristic. As the problem dimensions are increased the computational effort increases considerably. The computational time increases faster for randomly chosen dispatch rule than for random heuristic. Despite the increase in computational time, when using randomly chosen dispatch rules, the difference in results were not significant, leading to the conclusion that the random heuristic should be preferred.

Table 6.13: Average computational time for solving one problem.

Problem	Method	Seconds
6×6	Random Heuristic	1.05
	Random Dispatch Rule	2.73
10×10	Random Heuristic	9.23
	Random Dispatch Rule	38.72
14×14	Random Heuristic	36.15
	Random Dispatch Rule	191.76

6.3.1 Results of Other Research

In Runarsson et al. (2011) experiments are made with the Pilot method of Duin and Voß (1999) (equivalent to rollout algorithm of Bertsekas et al. (1997) discussed in chapter 3) as well as a Monte Carlo tree search algorithm on the same set of problems as here. Meloni et al. (2004) test the Pilot method on some well known 10×10 benchmark problems. The results of both papers are similar even tough they are not tested on the same set of problems.

The approach of setting the problem up in the framework of bandit algorithms, as is taken here, finds considerably better schedules than the papers mentioned above. The quantile rollout algorithm manages to decrease the average error of the 10×10 problems of up to 5% compared to the best results obtained with the Pilot method. For larger problems the gain is even more. The results are directly comparable to those of Runarsson et al. since they are tested on the same set of problems but since Meloni et al. test on different problem instances the results are not directly

comparable.

6.4 Summary

In this chapter the experimental setup was explained as well as the performance measures. The results for experiments with the fortified, average, hybrid and quantile rollout algorithms were presented for four different problems sizes and the computational effort discussed. For the larger problems the quantile rollout algorithm gave the best performance. The final chapters follows up on those results with summary and discussion.

7 Summary and Discussion

The rollout algorithms presented in this study show promising results and by using simple exploration/exploitation tactics it is possible to generate good schedules for job-shop problems.

For smaller problems, 6×6, the fortified rollout algorithm finds the optimal solution for most runs with a budget of 10.000 rollouts, especially with threshold ascent exploration policy. For the larger problems, 14×14 and 20×20, the rollout algorithms were able to generate good schedules in a reasonable time. For these problems instances the average rollout algorithm outperformed the fortified rollout and the hybrid rollout algorithm outperformed the average rollout. This is interesting since the fortified rollout algorithm is sequentially improving, but the average and hybrid rollout algorithms are not.

This can be explained by the fact that for larger problems the number of feasible solutions is extremely high and the probability of finding a good schedule early on is small. This may result in the fortified rollout algorithm putting too much emphasis on schedules found early in the scheduling process. The average rollout algorithm on the other hand dispatches the job that finds the best schedules on average and therefore does not depend upon finding good schedules early on. This generates promising positions in the solution space from where the fortified rollout algorithm is more effective than the average rollout, explaining why the hybrid rollout algorithm outperforms the average rollout.

For larger problem instances the quantile rollout algorithm gave the best results. The quantile rollout algorithm dispatches jobs that have the best average of good solutions, compared to the job with the highest average of total solutions as done by the average rollout algorithm. For the larger problems this method outperformed both the average and hybrid rollout algorithm. The reason is that when a job is dispatched based on the average there may be a greater probability of finding low quality solutions. Similarly, dispatches based on extreme values, such as the fortified rollout algorithm, may put too much bias on the best schedule found for the given set of rollouts, which in return may result in a low probability of visiting regions of greater solution quality. For the smaller problems the quantile rollout algorithm also gave very promising results even if the fortified rollout algorithm works better for these problems. The quantile rollout algorithm was the most consistent method,

giving good result for all problem sizes.

When using the average rollout algorithm the best exploration policy is to evenly distribute the rollouts between jobs. This gives us the best estimate of the true value of each job and our final choice is more reliable. When using the fortified rollout algorithm, threshold ascent is the best exploration policy since it allocates more rollouts to the more promising jobs and is therefore more likely to find good schedules.

Rollouts were made for six different rollout budgets: 100, 500, 1.000, 3.000, 5.000 and 10.000. There is no need for more than 3000 rollouts when solving smaller problem instances, but for larger problem instances (14×14 and 20×20) it might be possible to generate better schedules by increasing the rollout budget above 10.000. This will on the other hand, of course increase the computational effort of the algorithms.

Rollout algorithms are effective methods for finding approximate schedules for job-shop problems. But the computational effort increases considerably as the problem size is increased. The variable having the greatest impact on the computational time is the choice of pilot heuristic. Two different pilot heuristics were tested, the random heuristic and randomly chosen dispatch rules. When using randomly chosen dispatch rules, the computational time increased significantly. The average computational time needed for solving one problem of each problem size using randomly chosen dispatch rules was five times higher compared to using the random heuristic. Despite the increase in computational time the difference in results were not significant, leading to the conclusion that the random heuristic should be preferred.

8 Conclusion and Future Work

Rollout algorithms gave encouraging results for the job-shop problem. Within reasonable computational time promising and good schedules can be found. The best way to tackle smaller problem instances is to use the fortified rollout algorithm and dispatch based on the best overall schedule found from rollouts. However, for larger problems different strategies are needed.

The quantile rollout algorithm was the most successful rollout algorithm for solving larger job-shop problems. This method was also the most consistent one, producing good schedules for both the smaller as well as the larger problems. The advantage of the quantile rollout algorithm is that it dispatches jobs with a higher likelihood of finding good solutions. When a job is dispatched based on the average there may still be a greater probability of finding low quality solutions. Similarly, dispatches based on extreme values, such as the fortified rollout algorithm, may put too much bias on the best schedule found for the given set of rollouts, which in return may result in a low probability of visiting regions of greater solution quality.

The performance of the different rollouts algorithms is highly reliant on the underlying pilot heuristic used in the rollout phase. In the thesis two approaches were taken. The first was purely random job dispatching, this is the fastest way of performing rollouts. The second approach was significantly slower; this was the approach of dispatching according to a randomly chosen classical job-shop dispatching rule. These were, SPT, PL, MRT, LQUE, NM, RT, ST, ECT, LPT, FCFS (detailed descriptions of these dispatching rules may be found in table 2.1). Despite the increase in computational time the difference in results between the two pilot heuristics were not significant, leading to the conclusion that the random heuristic should be preferred.

Despite that a more effective pilot heuristic will undeniably lead to better schedules, but as the chosen pilot heuristic becomes more effective, it's computational cost might follow suit. Future work should aim at finding pilot heuristics that are able to find better schedules while being computationally efficient.

The rollout algorithms gave good results for deterministic problems but the methodology is easily transferrable to stochastic problems. Further work should go into discovering the applicability of these rollout algorithms to stochastic problems.

Bibliography

Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2nd edition.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine learning*, 47(2):235–256.

Banharnsakun, A., Sirinaovakul, B., and Achalakul, T. (2011). Job Shop Scheduling with the Best-so-far ABC. *Engineering Applications of Artificial Intelligence*, (2006):1–11.

Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C. (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3:245–262.

Brucker, P. (2007). *Scheduling algorithms*. Springer, 5th edition.

Brügmann, B. (1993). Monte carlo go. *White paper*.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Cicirello, V. and Smith, S. (2005). The max k-armed bandit: A new model of exploration applied to search heuristic selection. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20. AAAI Press.

Croce, F. D., Tadei, R., and Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers and Operations Research*, 22(1):15–24.

Duin, C. and Voß, S. (1999). The Pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks*, 34(3):181–191.

Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. F. and Thompson, G. L., editors, *Industrial scheduling*, chapter 15, pages 225–251. Prentice Hall.

Garey, M., Johnson, D., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129.

Hefetz, N. and Adiri, I. (1982). An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. *Mathematics of Operations Research*, 7(3):354–360.

Jain, A. and Meeran, S. (1998). A state-of-the-art review of job-shop scheduling techniques.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

Kawai, T. and Fujimoto, Y. (2005). An efficient combination of dispatch rules for job-shop scheduling problem. *IEEE International Conference on Industrial Informatics*, pages 484–488.

Laarhoven, P. J. M. v., Aarts, E. H. L., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125.

Meloni, C., Pacciarelli, D., and Pranzo, M. (2004). A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131(1):215–235.

Pan, C.-H. (1997). A study of integer programming formulations for scheduling problems. *International journal of systems science*, 28(1):33–41.

Pan, J. C.-H. and Chen, J.-S. (2005). Mixed binary integer programming formulations for the reentrant job shop scheduling problem. *Computers & Operations Research*, 32(5):1197–1212.

Pezzella, F. and Merelli, E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2):297–310.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 3rd edition.

Runarsson, T. P., Schoenauer, M., and Sebag, M. (2011). Pilot, Rollout and Monte Carlo Tree Search Methods for Job Shop Scheduling.

Streeter, M. and Smith, S. (2006). A simple distribution-free approach to the max k-armed bandit problem. *Principles and Practice of Constraint Programming*, 4204(2006):560–574.

Sun, D., Batta, R., and Lin, L. (1995). Effective job shop scheduling through active chain manipulation. *Computers and Operations Research*, 22(2):159–172.

Sutton, R. and Barto, A. (1998). *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Vaessens, R. J. M., Aarts, E., and Lenstra, J. (1994). Job shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317.

Yamada, T. and Nakano, R. (1992). A genetic algorithm applicable to large-scale job shop Problems. In *Parallel Problem Solving from Nature*, pages 281–290.